架构师

ARCHITECT

7月刊

推荐文章

阿里 P10、腾讯 T4、华为 18，
互联网公司职级、薪资、股权大揭秘

Geekbang. 极客邦科技 | InfoQ

CONTENTS / 目录

架构师

2019 年 7 月刊

本期主编 赵钰莹

流程编辑 丁晓昀

发行人 霍泰稳

提供反馈 feedback@geekbang.com

商务合作 hezuo@geekbang.org

内容合作 editors@geekbang.com

卷首语

2019 年，是属于容器技术的时代

作者 阿里巴巴高级技术专家 张磊

最 2019 年，全世界的开发人员都开始习惯用容器测试软件，用容器做线上发布，开始对容器化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。

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？

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。

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。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。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”。

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。

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

在 2019 年这个软件交付已经被 Kubernetes 和容器重新定义的时间点上，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。

我们能够看到，整个云计算生态正在尝试重新思考 PaaS 的故事。我们还能够看到，云的边界正在被技术和开源迅速的抹平。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。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。

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。

华为鸿蒙或将提前推出，谷歌被曝希望与华为继续合作

作者 赵钰莹 ，张晓楠

一昨天，环球时报发出推文表示：华为正在加紧测试自己的操作系统，国内市场命名为“鸿蒙”，海外市场命名为“OaK OS”，可能在 8 月或 9 月推出。一方面这使得余承东此前透露的“最快今年秋天，最晚明年春天”大大提前，另外也让谷歌的态度微妙起来。

根据英国《金融时报》最新消息，谷歌正在试图向美国政府表明：如果从美国国家安全的角度考虑，希望可以继续向华为提供新技术。

事件回溯

昨天，环球时报发出推文表示：华为正在加紧测试自己的操作系统，国内市场命名为“鸿蒙”，海外市场命名为“OaK OS”，可能在 8 月或 9 月推出。

这个时间，与此前余承东透露的“最快今年秋天，最晚明年春天”相比，提前了，有网友疾呼“中国自研操作系统即将到来。”而谷歌对此的态度也很微妙。

根据多家外媒的最新消息，谷歌正在试图向美国政府证明，需要能够以美国国家安全的名义向华为提供技术。关于这一举动背后的原因，有消息人士透露是因为谷歌停止与华为合作后，华为被迫将 Android 分为“混合”版本，这种版本将“更容易被黑客攻击，尤其是在中国”，谷歌希望保护其

用户在美国和全球数百万现有华为手机上的安全。

此外，谷歌相关发言人也被曝出“尤其担心不允许华为更新 Android 操作系统后，华为将会开发自己的操作系统”。相关行业专家表示，这些禁令短期来看可能会损害华为业务，但长远来看可能会迫使该公司及其他中国公司通过开发更多本土技术自力更生，从而损害谷歌等美国公司的主导地位。

谷歌态度反复变化：皆是因安全？

在这场拉锯战中，谷歌是最先明确表示与华为中止业务合作的企业，而停止合作的原因很重要的一条是“国家安全”。5 月 15 日，特朗普签署行政令，要求美国进入紧急状态，美国企业不得使用对国家安全构成风险的企业所生产的电信设备。当然，很多网友当时对谷歌的做法表示理解，认为其主要是受到政治因素的影响。

今时今日，谷歌以同样的理由要求美国政府重新考虑这一禁令，并认为停止与华为合作将会对国家安全造成影响。那么，脱离谷歌之后的 Android 操作系统是否确实存在无法弥补的安全问题呢？

此前，在谷歌宣布中止与华为的合作之后，华为曾发表过相关声明，表示：

> 安卓作为智能手机操作系统，一直是开源的，华为作为重要的参与者，为安卓的发展和壮大做出了非常重要的贡献。华为有能力继续发展和使用安卓生态…未来，华为仍将持续打造安全、可持续发展的全场景智慧生态，为用户提供更好的服务。

从声明中不难看出，华为希望现在以及未来可以持续为用户提供安全、可持续发展的安卓生态。并且，从目前曝光的信息来看，华为已经在加紧进行“鸿蒙”操作系统的研发。据透露，新款操作系统打通了手机、电脑、平板、电视、汽车和智能穿戴等设备，统一成一个操作系统，兼容全部安卓应用和所有 Web 应用；如果安卓应用重新编译，在这套操作系统上，运行性能提升超过 60%，这是面向未来的微内核。

对此，有知乎网友表示，脱离谷歌后的安卓操作系统是否可以保证安全性，可以参照国内安卓操作系统的发展过程。国外大部分安卓用户习惯去谷歌官方的应用市场 GooglePlay 下载，管控到位，APP 滥用权限的情况不多见。但是在国内，由于谷歌没在中国大陆开展业务，用户只能自己找下载渠道，形形色色的软件分发网站出现，碎片化问题、漏洞修补问题层出不穷。

过去数年，国内安卓手机厂商在安全方面做了不少努力，比如相关安全机构会在发现问题的第一时间同时抄送谷歌和对应的手机厂商，帮助手机厂商制定防御方案，缩短漏洞修复时间。因此，国内手机厂商在安全方面已经积累了不少经验，这种经验有希望被复用到其他市场。

因此，很多用户认为，安全并不是促使谷歌态度反复无常的唯一原因，这其中还包括担忧 Android 系统出现其他强大的分支版本，以及如果不能再和华为这样的大公司做生意，可能将损失大量资金，这一点同样体现在其他美国公司身上，比如高通，也被曝正在与美国政府洽谈。

目前，这一事件的结局尚不明晰，谷歌已经向华为提供了临时许可，这一许可将于 8 月 19 日到期，如果届时没有更新动态，华为方面应该会有所动作。

“鸿蒙”的挑战：生态！

对于“鸿蒙”可能的提前以及谷歌的态度，虽然很多人非常激动，但还是有很多网友还是保持了冷静和客观。

有人说：谷歌的操作系统一开始就是几个人开发的，苹果的 iOS 也是开发了三五年，所以华为准备了这么久也不是开发不出来，只是应用生态上可能需要很长的时间。

还有人说：华为虽然有自己的应用市场，但是不一定可以做到像谷歌那样好…

这其实说到了问题的关键，也是“鸿蒙”将面临的最大挑战：生态的挑战。

此前任正非在接受记者采访时提到过：做一个操作系统的技术难度不大，难度大的是生态，怎么建立起一个生态？这是一个大事情，慢慢来。

如果现在形势所迫，让“鸿蒙”没办法按照之前余承东透露的日子“最快今年秋天，最晚明年春天”的节奏慢慢来，那么华为准备好应对生态的挑战了吗？

根据曝光的“鸿蒙”截图，其 UI 设计、系统逻辑以及 App 安装界面与现在华为手机上的 EMUI 并没有明显区别，这会使得 EMUI 用户可以尽快习惯新系统，降低学习成本。但是已经有 EMUI 用户吐槽 EMUI 并不好用，这将对“鸿蒙”在认知上带来挑战。实际上“鸿蒙”系统与基于安卓的 EMUI 毕竟是两个完全不同的系统，把”鸿蒙“和 EMUI 混淆在一起真的好吗？

其实一个系统再优秀，如果没有足够的软件支持、开发者支持也是不行的。“鸿蒙”操作系统界面之外，也希望华为开发工具、开发文档能够按时间节奏发布，在生态系统建设上有更进一步的积极行动。

不过从另外角度来看，2018 年华为手机出货量已经达到 2 亿台以上，加上操作系统打通手机、电脑、平板、电视、汽车、智能穿戴后，华为在其他智能硬件方面的支持，这样的用户生态，也具有威慑力。

联想到不久前华为成立智能汽车解决方案 BU，汽车作为继手机、电脑后的最重要终端设备，华为在汽车上的布局，也是壮大智能硬件生态的一步重要棋局吧。

这里记者还注意到来自 Huawei Central 的 APP 系统截图中，提到了“安卓绿色联盟”。安卓绿色联盟是 2016 年 11 月 14 日由华为主导，联合阿里巴巴、百度、腾讯、网易四家企业共同发起的。五家创始企业涉及的应用达 500 多个，涵盖众多主流移动应用，几乎覆盖所有国内终端用户。如果该截图属实，鸿蒙系统在日常使用和国内应用生态上也会比较顺利。

交付程序不给钱，程序员一怒之下开源客户项目代码

作者 小智

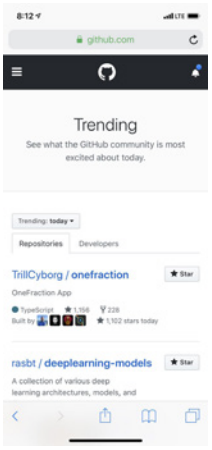
国外一名自由职业的开发者在客户不给结款以后，将其开发的项目开源到了 GitHub 上，两天不到收获了超过 3000 个 star。除了此事引发的技术细节大讨论以外，背后问题不得不引人深思：开发者的职业道德与法律上的漏洞引发的事故，该如何看待？

不给钱就开源

最近，外国一位名叫 Jason Werner 的程序员小哥过得很闹心。

事件的起因是自由职业的他接了个外包需求，给一位客户开发一个名为 OneFraction 的平台。在跟客户沟通好需求后，他开始写这个项目，结果项目完成后客户摆了他一道，最后也没给他钱。于是他一怒之下把这个平台开源到了 GitHub 上，短短两天时间不到，这个项目的 star 数就已经超过了 3000 个，更是登上了 GitHub 的 Trending 榜单的榜首。项目地址。

"I always wanted to be at the top of trending."



按照 Jason 小哥的说法，OneFraction 的设计初衷是作为一个平台，让用户通过平台支付租金而不是支票或银行转账。其价值来自于利用平台产生的数据最终创建一个租赁市场，用户可以通过其找到完美的公寓。

他的技术选型是这样的：Server 端用 Node.js 写就。服务器使用 GraphQL 和 apollo-server 在客户端和服务器之间传递数据，并以类型友好的方

式键入以与 Mongo 交互。账户系统则选用 accounts.js。

Jason 的情况在外网引起了热议，在队形表达对 Jason 的同情以后，大家愉快地聊起了 OneFraction 背后的技术细节，不得不说中外程序员们一大相同点就是对技术的热情和兴趣，一群可爱的技术宅。

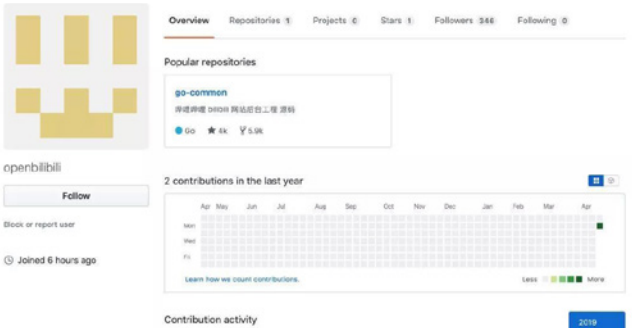
这样一个闹心的事件逐渐演变成了一场有关技术的 Party 让人忍俊不禁，但这背后的一些问题却值得我们深思。Jason 没有取得相应的报酬，对其创造的代码自然有全权处理的资格，但如果取得了报酬或者部分报酬，这样的问题该如何处置？当开发者面对道德选择与法律困境时，又该作何取舍？

这事儿是职业道德问题？

跟 Jason 相类似的情况在国内也有发生过，但更多的是一些私自开源公司代码、删库跑路的相关新闻，这背后，折射的是开发者的职业道德问题吗？

程序员私自开源公司代码，如何定责？

还记得不久前的哔哩哔哩后端源码泄露事件吗？



上线 6 小时 star 数突破 5 千，fork 数一路狂飙冲过 6 千，知乎、微博、朋友圈等各大程序员聚集地都在讨论这个相同的问题：究竟是主动开源还是被黑了？

后来官方回应确认了这是事故，不是故事。

源码的重要性不言而喻，有了源码，黑产不用再通过逆向工程的手段猜测运行原理和漏洞位置，直接通过阅读源码就可以找到很多还未公开的漏洞，这对系统的安全性是个巨大的威胁。不管源码发布者是何人、出于何种目的，这种行为本身既无职业道德，也触犯了法律。

根据极客时间《白话法律 36 讲》专栏作者周甲德律师的观点：

> 这要看程序员是不是具有主观故意，如果具有主观故意，轻则可能会被开除，并赔偿公司损失；重则可能构成刑事犯罪，代码在公司没有开源前应当属于公司的商业秘密，根据我国《刑法》第 291 条之规定，其可能涉嫌侵犯商业秘密犯罪。这里面主要是看此行为给公司造成了什么后果，如果给公司带来的影响小损失不大，公司可根据公司规定进行处理。如果程序员只是失误失职，没有主观故意，则可能会受到公司内部处理。

删库跑路，当笑谈成为新闻

“rm-rf”应该是你非常熟悉的命令行，也是删库跑路梗常用的背锅侠。但其实删库跑路这样的事情，真的发生过很多次。荷兰一家云主机厂商 Verelox，就曾被离职工程师删除了客户数据库，几经修复也不能完全恢复，最后造成巨大损失的。

新华社之前也曾报导过类似的新闻，某软件工程师徐某，离职后半年公司仍然不给结清工资，于是其一气之下，利用自己当年开发网站写代码时留下的后门文件，把程序源码全部删除，造成几十万损失。

这样恶意删库的行为，触犯了破坏计算机信息系统罪，将要受到法律的制裁。

> 本部分案例援引自极客时间《白话法律 36 讲》专栏

开发者是保护代码道德的最后防线?

2018 年 3 月，Stack Overflow 发布了他们的开发者调查报告，并首次提出了有关道德的问题。对于“开发人员是否有义务考虑代码的道德影响”这个问题，有近 80 % 的人回答“是”。不过，只有 20 % 的人认为他们最终在为不道德的代码负责，40 % 的人会在被要求的情况下写不道德的代码，只有 50 % 的人表示在发现不道德的代码时会举报。

如果代码对世界的影响不大，那么这也许就不成问题。打个比方，如果你写了一个对 100 个人不利的算法，虽然这事不怎么光彩，但产生的影响也是有限的。但是，如果你在拥有数亿用户的 Facebook、Google、微信上做同样的事情，结果就会很严重。

但对于开发者来说，光是每天写业务代码就已经让人心力交瘁了。更何况不管在国内还是国外，技术在大部分时候都是为业务服务，开发者的

话语权是拗不过盈利的这条大腿的。遵守程序员的职业道德已经不容易，又如何去捍卫代码的道德?

开发者：我讲道德，谁讲法律？

当出现 Jason 这样的案例时，开发者们无一不在拍手称快：”四个字，大快人心！“

为什么在程序员们看来，会出现这样的现象呢？究其根本，无非就是开发者和用人单位在一个不对等的位置上。

> 曾几何时，在一次工作交接的时候，因为我的项目由我全权负责和开发维护，公司对我就像防贼一样。

企业在聘用程序员以后，又或多或少地在某些地方对程序员处处提防，当然这本无可厚非，但也体现了劳资双方的不平等和不信任。尤其是在当下这个就业难的大环境下，开发者们敢于发声的底气便更加微弱。

开发者们坚守着自己的职业道德，却也害怕遇上不讲法律或者钻法律漏洞的用人单位。个体的力量终归有限，时间、精力都是成本，拖家带口的开发者根本耗不起。

> 如果是你，在面对 Jason 遭遇的问题时，会怎么处理？在你看来，开发者在日常工作中应该注意的法律常识有哪些？

改进 TCP，阿里提出高速云网络拥塞控制协议 HPCC

作者 赵钰莹

TCP 是最基础的网络传输层通信协议，其拥塞控制算法是为 Internet 这种相对低速、高延迟的网络环境设计的。在新一代的高速云网络中，TCP 的拥塞控制算法无法充分发挥底层网络能力，而现有高速网络拥塞控制算法（如：RDMA 协议中的拥塞控制算法 DCQCN）又存在严重的稳定性风险。阿里巴巴的技术人员研发了新一代高速云网络拥塞控制协议 HPCC（High Precision Congestion Control），旨在同时实现高速云网络的极致性能和超高稳定性。目前这一成果已被计算机网络方向世界顶级学术会议 ACM SIGCOMM 2019 收录，引起了国内外广泛关注。

根据 IETF 的 RFC 793 定义，TCP 是一种面向连接的、可靠的、基于字节流的传输层通信协议。在过去的几十年中，TCP 相关的基本操作未做太大改动，但其拥塞控制算法经历了几代学者和工程师的迭代更新，比如 RFC2581《TCP 的拥塞控制》、加州理工学院研发的 FAST TCP 以及目前在数据中心网络中被广泛使用的 DCTCP 等。

然而，在过去几十年中发生变化的不仅仅是算法本身，网络环境也发生了巨大变化，尤其是云计算出现之后，云租户和应用对网络带宽、延迟以及稳定性的要求比过去的互联网用户提升了一到两个数量级，这导致传统 TCP 协议开始难以适用于云网络（数据中心网络）的高速、低延迟环境。虽然 RDMA 等新一代技术摒弃了传统 TCP 中的一些做法，以换取网络性能的大幅提升，但是 RDMA 拥塞控制算法本身蕴含着相当高的不稳定性风险，阻碍其在大规模网络中得到广泛应用。阿里巴巴的一群工程师近期成功研发出新一代、超高性能云网络环境下对传统 TCP 和 RDMA 拥

塞控制算法的替代方案 –HPCC，InfoQ 第一时间对 HPCC 团队成员阿里云智能研究员张铭、阿里云智能高级技术专家刘洪强进行了独家专访，探究 HPCC 的研发背景及实际意义。

HPCC (High Precision Congestion Control- 高精度拥塞控制)

随着云计算的迅猛发展，传统 PC 时代的小型机房的网络架构已逐渐退出历史舞台，取而代之的是以数据中心为核心的超大规模、云网络架构。在这样的环境下，目前主流的 TCP 和 RDMA 拥塞控制算法（例如 DCTCP,DCQCN）要么无法充分发挥云网络低延时、高带宽的优势，要么无法在大规模网络环境下保持稳定。这给包括阿里巴巴在内的大型云计算服务商们带来了严峻的运营和技术挑战。

HPCC 是在高性能的云网络环境下，对现有的拥塞控制的一种替代方案。它可让数据中心网络中的报文稳定的、以微秒级的延迟传输。当前主流的拥塞控制算法主要依赖于端的信息（例如丢包信息，延迟信息），以及极为有限的设备反馈信息（如 1 个比特的 ECN）做拥塞控制，而 HPCC 则创新性地运用了最新网络设备提供的细粒度负载信息而全新设计了拥塞控制算法。在 HPCC 的帮助下，主流的云应用，比如分布式存储、大规模机器学习，高性能计算等性能会得到几倍到几十倍不等的提升；云租户相应地将会感受到延迟显著降低，效率和性价比大幅提升。

核心：拥塞控制

无论是 TCP、RDMA 还是其各种改进版本，其核心都在围绕拥塞控制算法进行，这也是高性能云网络中必须解决的痛点，而 HPCC 的核心就是重新定义下一代拥塞控制机制。张铭表示，HPCC 的思路和框架同样可以用于改进 TCP 或者 RDMA 等其他传输层协议。

在计算机网络里，传统的拥塞控制算法主要通过在端上调节流量，以维持网络最佳平衡状态。发送方根据网络承载情况控制发送速率，以获取高性能并避免拥塞崩溃（congestion collapse）导致网络性能下降几个数量级，并在多个数据流之间产生近似最大化最小流的公平分配。发送方与接收方确认包、包丢失以及定时器情况，估计网络拥塞状态，从而调节数据流的发送速率，这被称为网络拥塞控制。

HPCC 的核心理念是利用精确链路负载信息直接计算合适的发送速率，而不是像现有的 TCP 和 RDMA 拥塞控制算法那样迭代探索合适的速率；HPCC 速率更新由数据包的 ACK 驱动，而不是像 DCQCN 那样靠定时器驱动。

在整个研发过程中，刘洪强提到，最难的其实是思想和理念上的突破，拥塞控制算法已经被研究了数十年，只有突破传统的思维方式才可能得到新的收获；其次，完成这项研究需要同时聚齐懂理论、懂工程实践、懂硬件和懂运维的人，这本身就实属不易。最后，还需要攻克各种工程难关，将算法付诸实践。

当然，真正实现 HPCC 的大规模、商业化落地还需要一段时间，但阿里内部已经在模拟真实网络的实验环境下进行了多方面验证，其效果与设想高度一致。

经过各种软硬件的精巧设计，阿里工程团队已完整和高效地实现了 HPCC 的软硬件协议栈。实验表明，HPCC 在拥塞条件下可以将延迟降低一到两个数量级，且收敛速度极快；一旦出现空闲带宽，立刻会被充分利用，整体网络利用率维持在相当高的水平，而延迟则接近于理想值。张铭强调，在无拥塞的情况下，数据流的传输速度都很快；而一旦发生拥塞，受影响的数据流从不稳定状态恢复到稳定状态的时间需要越短越好，HPCC 的收敛速度和稳定性都要远优于目前的主流算法。

研究价值

目前业内对网络传输协议的选择基本分为两大类：一类是以 TCP 为主，持续探索如何将 TCP 的性能调至更优的状态；另一类则希望研究可以取代 TCP 的新传输协议。张铭解释道，HPCC 的出现为下一代拥塞控制开拓了一个全新的方向，无论是 TCP, 还是 RDMA，抑或是某种新的传输层协议，都可以直接使用 HPCC，或是在其基础上构建适用于高性能云网络的拥塞控制机制。

对云上租户而言，HPCC 的价值在于可以让其享受高速的网络服务，而不需要担心稳定性问题；网络资源利用率提升的同时会带来云使用成本的降低；而对性能要求极高的用户（如 HPC），则更是至关重要。

结束语

对基础科学的投入和支持不能只停留在理论层面，还需要紧密联合一线工程师与应用场景，也许这正是阿里巴巴成立达摩院支持基础研究的原因。正如阿里云智能总裁，达摩院院长张建锋在 2019 年阿里云峰会上所说:

> **阿里巴巴的科研力量将融会贯通。达摩院的能力将与云全面结合。未来还将加大研发投入，扩大云的技术代差优势。**

目前，该项研究的学术论文《HPCC: High Precision Congestion Control》已经被网络顶级学术会议 SIGCOMM 2019 录取，该论文详细介绍了阿里巴巴自研的新一代高速网络拥塞控制协议。

“现代数据中心中数以万计的处理器相互之间的通信如何被组织在一起来避免他们之间通信通道的拥塞呢？这是我在杭州初次遇到这一群阿里巴巴专家时，他们正在尝试解决的问题。…我相信我们在这个领域已经做出了非常重要的进展，而且我们非常高兴我们的阶段性工作已经被 SIGCOMM 2019 接收”–HPCC 论文的合作作者，英国皇家科学院院士，剑桥大学 Frank Kelly 教授表示。

关于 SIGCOMM

SIGCOMM 是 ACM 组织在网络领域的旗舰型会议，也是目前国际网络领域的顶尖会议。几十年以来，多项经典研究成果都出自 SIGCOMM 大会，比如《Development of the Domain Name System》(SIGCOMM 1988)，阐述了互联网域名管理系统（DNS），这套系统已经被使用几十年，贯穿了互联网的发展史；《Congestion Control in IP/TCP Internetworks》(SIGCOMM 1987) 和《Congestion Avoidance and Control》(SIGCOMM 1988)，奠定了互联网 TCP 拥塞控制的基础，其算法设计思想一致沿用至今；《Ethan: Taking Control of the Enterprise》(SIGCOMM 2007)，软件定义网络 (SDN) 思想的开山之作，SDN 使得大规模网络虚拟化成为可能，让“云网络”的概念落地。

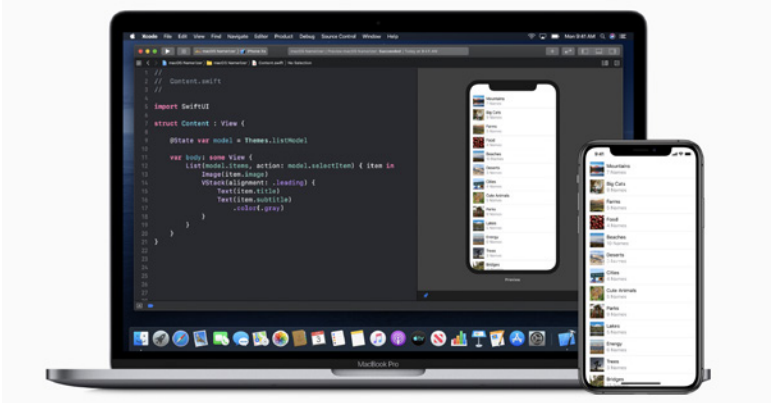
苹果发布全新 SwiftUI 框架：一次编码，五端通用

作者 Apple Developer
译者 王强

在 WWDC2019 上，Apple 推出了一个全新的 SwiftUI 框架，这是一个现代化的 UI 界面编码结构，它是从头开始构建的，以利用 Swift，让开发者感到惊讶。新框架使用声明性范例，让开发者用更少的代码编写相同的 UI。另外，SwiftUI 在 Xcode 中启用实时 UI 编程环境，实时看到编码的页面效果。最令人开发者尖叫的是，实现一次编码，可适应五端 Apple 产品平台。

应用更完美，代码更少

SwiftUI 是一种非常简单的创新方法，可以利用 Swift 的强大能力在所有苹果设备平台上构建用户界面。通过 SwiftUI，开发者仅使用一组工具和 API 就能为所有苹果设备构建用户界面。SwiftUI 使用易于阅读和编写的声明式 Swift 语法，可与新的 Xcode 设计工具无缝协作，使你的代码和设计完美同步。SwiftUI 自动支持动态类型、黑暗模式、本地化和可访问性，你的 SwiftUI 代码将成为你写过的最强大的 UI 代码。



声明式语法

SwiftUI 使用声明式语法，开发者可以简单地声明你的用户界面要做的事情。例如，你可以声明你要加入一个包含文本字段的项目列表，然后描述每个字段的对齐方式、字体和颜色。你的代码将比以前更简洁易读，节省你的时间并简化维护工作。

```
import SwiftUI

struct Content : View {

    @State var model = Themes.listModel

    var body: some View {
        List(model.items, action: model.selectItem) { item in
            Image(item.image)
            VStack(alignment: .leading) {
                Text(item.title)
                Text(item.subtitle)
                    .color(.gray)
            }
        }
    }
}
```

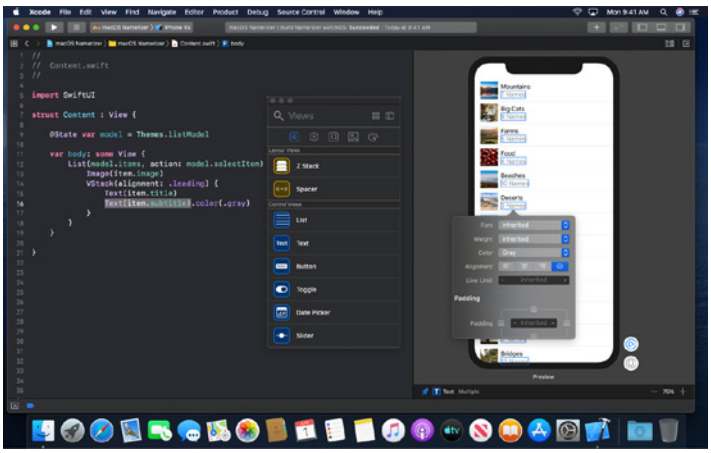
这种声明式风格甚至适用于像动画这样的复杂概念。只需几行代码就能轻松将动画添加到几乎任何控件中，并为其选择一系列现成效果。在运行时，系统会处理创建平滑移动所需的所有步骤，甚至可以处理中断以保持应用稳定。现在加入动画变得如此简单，你就能寻找新的方法让应用变得更加生动有趣。

设计工具

Xcode 11 提供了新的直观易用的设计工具，现在使用 SwiftUI 构建界面就像拖放一样简单自如。当你在设计画布中工作时，你编辑的所有内容都与相邻编辑器中的代码完全同步。键入代码时会即时显示预览效果，对该预览所做的任何更改都会立即反映在代码中。Xcode 会实时重新编译你的更改，并将其插入到应用的运行版本中，始终可见且可编辑。

拖放：只需拖动画布上的控件即可在用户界面中排列组件。单击打开检查器以选择字体、颜色、对齐方式和其他设计选项，并使用光标轻松重新排列控件。其中一些可视化编辑器也可以在代码编辑器中使用，因此就算你更喜欢手写部分界面代码，也可以使用检查器为每个控件发现新的修改器。你还可以从库中拖动控件并将其拖放到设计画布上或直接放在代码

上。



动态替换：Swift 编译器和运行时完全嵌入到 Xcode 中，因此你的应用将不断构建并运行。你看到的设计画布不仅仅是用户界面的模拟——它就是应用的实时效果。Xcode 可以直接在你的实时应用中使用“动态替换”交换编辑过的代码，这是 Swift 中的一项新功能。

预览：现在你可以为任何 SwiftUI 视图创建一个或多个预览，以获取示例数据并配置几乎所有用户可能看到的内容，诸如大字体、本地化或黑暗模式。预览还可以显示你的 UI 在任何设备和任何方向上的效果。

原生支持所有苹果设备平台

SwiftUI 基于苹果数十年来创建世界上最直观，最具创新的用户界面的经验打造。苹果生态系统上所有用户喜爱的内容，例如控件和平台专属体验等，都会在你的代码中优雅呈现出来。

SwiftUI 是真正的原生代码，因此你的应用程序可以通过少量代码和交互式设计画布直接访问每个平台的成熟技术。

上手体验

这里有一系列教程和文档，提供手把手指导和深度教学：

SwiftUI 教程：https://developer.apple.com/tutorials/swiftui/

SwiftUI 文档：https://developer.apple.com/documentation/swiftui/

阿里P10、腾讯T4、华为18，互联网公司职级、薪资、股权大揭秘

作者 小智

BAT、头条、华为们的最新技术职级序列是如何划分的？各个职级的大致薪资范围、股票都是多少？不同职级之间的晋升难度有多大，如何评审？校招、社招程序员想去大厂应该如何准备？本篇文章搜集整理了最新版数据，为你一一解答！

写在前面

相信很多读者朋友在网上看过不少互联网公司薪资的问题和数据报表，其中很多数据的更新时间都停留在2到3年前。InfoQ的编辑于是在想，关注这个问题的人如此之多，但信息却很久没有更新了，有没有可能我们去做一些相关的事情，让大家了解到最新的详情？

于是我们就去咨询了各大互联网公司的程序员们，一位低调路过的阿里P9回复我说：“绝对不能说！说了我就被开了！”。阿里这条线没有突破，腾讯那边也没有得到想要的信息：“职级、薪资这些都是敏感信息，不方便透露”。

条条大路通罗马，在职的程序员们不好说，那我们就去问已经离职的大厂员工，一位刚从百度离职飞往腾讯的姑娘这样跟我说：“入职的时候签了保密协议，离职的时候也签了保密协议，不能说啊。”

我们依旧不死心，又把目光瞄准了行业内的资深猎头。功夫不负有心人，这才有了今天文章的主体数据。需要声明的一点是：本文所有数字均不是官方数据，而是根据猎头接触到的候选人实际情况给出的参考数字，实际offer金额有可能更高，也有可能更低，但可以让你大致了解一个具

体范围。

BAT、头条、华为职级薪资报告

以 BAT 为代表的互联网公司，其职级规则已经成为整个行业的标杆，不管是 BAT 系的创业公司，还是成长中的各型企业，都在参照 BAT 的职级规范给公司员工评定级别、职称、薪资等。比如阿里的技术序列是 P 系列，腾讯、百度的技术序列是 T 系列，而华为则是数字系列。

阿里篇

阿里巴巴集团采用双序列职业发展体系，技术线就是常说的 P 序列，对应到管理线的 M 序列，P6 相当于 M1，P7 相当于 M2，以此类推。

层级	层级名称	层级	层级名称
		M10	董事长(chairman)
P14	资深科学家	M9	副董事长(Vice Chairman)
P13	科学家	M8	执行副总裁（EVP)
P12	资深研究员	M7	资深副总裁（Sr. VP)
P11	高级研究员	M6	副总裁（VP)
P10	研究员	M5	资深总监
P9	资深专家	M4	总监
P8	高级专家	M3	资深经理
P7	专家	M2	经理
P6	高级工程师	M1	主管
P5	中级工程师		
P4	初级工程师		

图片：阿里巴巴校园招聘

作为技术线的 P 序列，一共分为 14 级，从 P1 到 P14，目前校招过程中几乎不会涉及 P1-3 级，最低从 P4 开始。而据笔者的了解，目前阿里巴巴集团大部分的校招最低级别也是 P5、P6。

目前阿里需求量最大的职级范围分布在 P6-P8，这也是阿里集团占比最大的级别。P6 级别的程序员 title 是高级工程师，P7 便已经是专家级别，P8 则是高级专家。一般而言，江湖上行走小有名气的阿里程序员至少也是 P8 级别。P10 级别的存在就是传说中的大神级别，这个级别的程序员无一不是业界鼎鼎有名的存在，比如褚霸、毕玄等等。

InfoQ 搜集了阿里巴巴职级体系下的薪资水准和股数，具体参考下表:

级别	薪资（16薪）	股票 (股数, 4年总计)
P6	40W	几乎不授予
P7	50-70W	800-1200
P8	70-100W	2000-2200
P9	100-120W	6000-8000
P10	150W+	12000+

阿里巴巴员工的薪资结构一般是 16 薪，好的团队年终奖可以拿到更多。另外，随着时间的推移，阿里巴巴在薪资和股票这两块表现出了相反的势头，薪资涨幅比较大，而授予的股数则下降明显。据了解，大概 7 年前，一个阿里 P7 员工可拿到 2400-3200 股，而现在 P7 级别所授予的股数也就 800-1200 股，同样工作满 2 年才能拿，分 4 年拿完。

蚂蚁金服的薪资现金部分与阿里巴巴较为相似，但期权部分相对比较宽松，早期甚至能拿到 2-3 万股，现在 P7 级别也能拿 2000 股。但可以肯定的是，当蚂蚁金服成功上市以后，对新员工的股票授予也会相应下降。

腾讯篇

腾讯不久前刚刚宣布调整职级，取消了原有的 6 级 18 等（1.1-6.3 级）的职级体系设计，将专业职级体系优化为 14 级 (4-17 级)。

职级体系（旧）		职级体系（新）	
专业职级	专业title（技术研发通道为例）	专业职级	专业称谓（技术研发通道为例）
6.3 6.2 6.1	权威专家	17	17级工程师
5.3 5.2 5.1	资深专家工程师	16	16级工程师
		15	15级工程师
4.3 4.2 4.1	专家工程师	14	14级工程师
		13	13级工程师
		12	12级工程师
3.3 3.2 3.1	高级工程师	11	11级工程师
		10	10级工程师
		9	9级工程师
2.3 2.2 2.1	工程师	8	8级工程师
		7	7级工程师
		6	6级工程师
1.3 1.2 1.1	助理工程师	5	5级工程师
		4	4级工程师

在腾讯，技术线在此之前属于 T 序列，在腾讯的职级体系里，T3 级别已经是很多人的上限，行走江湖有名有号的 T4 级别更是当得起各技术分享大会的技术爱好者们一声老师的称呼。而 T5 级别在整个腾讯也是凤毛麟角，代表人物有玄武实验室的于旸、优图实验室的贾佳亚等。

InfoQ 搜集了腾讯新职级体系下的薪资水准和股票价值，具体参考下表。

值得一提的是，虽然在老职级体系下，整个 T3 序列的 title 都是高级工程师，但每个小职级范围之间的薪资差距并不小，T3-3 级别的薪资比 T3-1 级别要高出 30-60W/ 年，且 3-1 级别几乎没有股票。在调整成数字序列以后，这样的差距看起来会显得更加合理。

级别	薪资（16-18薪）	股票 (价值, 每年折合)
T3-1 / 9级	38-40W	几乎不授予
T3-2 / 10级	50-75W	15-50W
T3-3 / 11级	70W-100W	30-70W
T4-1 / 12级	120-160W	80-120W
T4-2 / 13级	200-300W	100W+

腾讯的薪资结构一般是 16 薪，但实际上从 offer 看不乏 18 薪的团队。腾讯内部不同事业线之间存在不小的薪资、股票差距，腾讯的游戏团队薪资、年终奖一般都比较高，而腾讯云的股票份额则要高于游戏团队。

百度篇

百度是整个 BAT 中现金给得最多的。

和腾讯相同，百度技术线也是 T 序列，T5、T6 是技术线占比最大的级别。一般而言，在百度 T5 是高级工程师、T6 是资深工程师，但实际上百度的 title 并没有职级重要。从 T7 级别开始，就开始要做带团队、做管理的事情，升到 T7 以上后基本就不做写代码的事情了。T10-T12 的人数非常少，具有代表性的人物有前百度首席科学家吴恩达、百度最年轻 T10 楼天城等。

InfoQ 搜集了百度职级体系下的薪资水准和股票价值，具体参考下表:

级别	薪资（14.6薪）	股票 (价值, 每年折合)
T5	30-45W	几乎不授予
T6	40-60W	20-30W
T7	55-80W	40-60W
T8	75-100W	60-80W
T9	100-150W	100W+

百度的薪资结构是 14.6 薪，其特点是薪资的现金部分在 BAT 三家中是最多的。

华为篇

严格意义上来说，华为不算互联网公司，网上的职级、薪资数据也没有对华为进行过调查、报导。InfoQ 为此特意了解了华为背后技术线的薪资体系，以供参考。

华为技术线的职级体系为数字序列，跟腾讯的新序列相近。华为有句俗语很好地描述了收入情况：三年一小坎，五年一大坎。意思是入职华为

三年内大部分靠工资，三年后奖金逐步可观，五年后分红逐步可观。事实上，根据 InfoQ 调查了解到的情况也确实如此，在华为供职年限越久，奖金越多，分红规模越大。2015 年，现任华为高级副总裁陈黎芳在北大宣讲时提到：奋斗越久越划算，工资变成零花钱。

InfoQ 搜集了华为职级体系下的薪资水准和股票价值，具体参考下表:

级别	薪资 (每年总包)	虚拟股
16	40-60W	无
17	60-80W	跨度很大，建议参考薪资总包
18	100W-180W	
19	180W+	

在华为内部，除了薪资之外，奖金规模也不遑多让，这其中尤以终端部门的奖金为多。另外，华为公司内部还有一个名为 TUP 的虚拟股:

按华为《2015 年虚拟受限股分红预通知》，每股分红 1.95 元，升值 0.91 元，合计 2.86 元，工作五年基本可达十五级，饱和配股（包括 TUP）9 万股，分红 + 升值达 2.86*9 万 =25.74 万元，即使不饱和配股，基本分红也可以达到税前 20 万。工作 10 年，17 级配股普遍超过 20 万，税前分红 + 升值超过 50 万，而 23 级虚拟股票超过 200 万股，税前分红 + 升值超 500 万。（数据仅供参考）

华为每年的分红收益并不固定，2013 年度每股分红 1.47 元，2014 年度每股分红 1.90 元，2015 年度每股分红 1.95 元，2016 年度每股分红 1.53 元，2017 年度每股分红 1.02 元。虽然每年的收益并不稳定，但这对于华为员工来说却已经是让外人眼红的福利了。

头条篇

> InfoQ 采访的猎头说：头条的职级体系我们猎头一般不会作为参考。一般来说，头条的现金薪酬要比 BAT 们高出 25%-40%，同样是 16 薪的薪资结构。跳槽去头条的更多关注的是现金薪酬，而不是职级。

所以，有头条的同学愿意匿名分享一下吗?

BAT 内部技术晋升有多难？

虽然 BAT 们的技术岗位定着高大上的职级、拿着令人艳羡的薪资，但其实很多人在很多年里都会困在某一职级上停步不前，工作经验的积累并不能带来工作上的平滑晋升。

以阿里巴巴技术岗为例，很多人入职时可以拿到 P5、P6 的定级，但从 P6 到 P7 升级是一个坎，很多人会卡在 P6 级别上一两年甚至更久。而从 P7 到 P8 就更不容易了。再往上，从 P8 到 P9 的升级会更难，要的不仅是业界影响力，还需要有足够的运气。而从 P9 到 P10，难度更上一层楼，猎头直言：“这个级别需要做出像钉钉、咸鱼式的产品才有机会”。当成功晋升 P10 时，已经是管理线的 M5 级别，有机会进入阿里组织部，这个级别的技术人跳槽就很少了，一般都是出去创业。

腾讯的技术晋升也不容易。在腾讯旧的职级体系下，T3-3 升 T4-1 是一个大坎，停留在 T3-3 超过 5 年的不在少数，停留 7 年的也有。一旦进入 T4 级别，就是腾讯的专家工程师了，腾讯研发人数将近 2 万人，T4 级别的人数大概也不超过 500 人，这还是在近两年 T3 到 T4 级别人数增多的情况下。

百度技术晋升的第一个坎在 T5 到 T6，越往上越难。但对比之下，百度的技术晋升稍微容易一些。按照猎头的说法，百度即便是高层的晋升都比较平滑，没有大的过错、失误，一般都能顺利晋升。

工程师的晋升方式，不同公司之间的规则不尽相同。以阿里为例，每年 4 月份会组织一次工程师答辩。评委会由阿里技术线的高级程序员组成，对绩效考核达到 3.75 的员工进行考核答辩，通过者方能成功晋级。当然，如果你能做出惊天地泣鬼神的产品，跳级晋升也不是什么难事。

基本功扎实、技术能力过硬，这是技术线较低级别晋升的共性。但当发展到中、高级技术路线时，技术能力就不再是唯一重要的考核标准，具不具备良好的产品感觉、做没做过完整的技术架构、懂不懂业务痛点、商业思维，都是晋升必不可少的要素。

程序员去大厂，如何准备？

很多程序员、计算机专业的学生，对于挂着金字招牌的互联网大厂们都有一颗虽不能至，心向往之的心。InfoQ 采访的业界知名猎头 Denny 建议：

程序员去大厂，应该分人和阶段。在校招阶段，毕业生们尽量去大厂比较核心的部门，核心部门资源多、成长快，比如阿里巴巴的钉钉、阿里云，腾讯的腾讯云、游戏等部门。

如果有一到两年的工作经验，想跳槽去大厂，建议以一张白纸的身份过去，拥抱变化、主动学习，这样获得的成长可能比之前一到两年更加多。

很多年轻程序员在工作中最容易犯的错不是技术上的错，而是思维模式上的局限。归根结底，中国大部分互联网公司仍旧是业务驱动的模式，技术是业务发展背后的有力支持，但很少有技术驱动业务的模式。所以年轻程序员，一定要培养产品意识，主动去了解业务，这样才能从单纯的 Crud Boy 晋升成中、高级技术人乃至技术管理者。

在做打算的时候，想清楚自己要的是什么。单纯从薪资的角度讲，BAT 已经不是变现最快的选择。去大厂可以有好的资源、比较高的职级，甚至有收拾一个烂摊子脱颖而出的机会。而在创业公司，什么都缺，多面手更加吃香。等到准备面试的时候，要做好几手准备：

- 扎实的基本功，面向搜索引擎编程在工作中可以，但在面试中一定要越懂细节越好；
- 清楚面试岗位的需求，针对性地下功夫补强短板、提炼亮点；
- 充分了解自己，明白自己的能力边界，简历中轻易不要写我主导、我精通、我负责……；
- 锻炼沟通能力，良好的沟通能力在面试中是一个极大的加分项。

技术 leader 怎么看职级？

InfoQ 采访了两位技术出身的 CEO&CTO，问了问他们怎么看待职级和薪资这些问题。

- 贝壳金服 CEO 孔令欣

对于职级，我在乎也不在乎，主要看面试者从哪儿来。如果人选来自一些大公司，他的职级可能是一个参考值。但是一些小公司的职级比较乱，我都招过原来做过 CEO 的人跑到我们这边来做个总监，甚至只是做 VP 或者专员。

所以一般薪资的对比反而会更精准，他值多少钱，用这个价钱就再去市场上寻找相应的职级位置，当然有些人他可能会过高或者过低，但是不管怎么样，薪资的参考比职级的参考其实更有用。

我招聘看重什么？一是聪明，聪明不只是学习能力强，而是他自我迭代能力强，能不能在受到挫折或者是压力的情况下，去接受意见、自我迭代。这些通过面试、一些面试题是可以问出来的；

二是道德要好，我们是做金融的，所以更看重这一点。这个人如果很

自私，道德理念摇摆都不行。我们也有一些针对性的面试题和文化题，去把关道德方向。

但是很多时候我更看重一个人的成长力，潜力其实是很重要的一件事情，如果你有办法识别有潜力的人并把他招进公司，他跟公司一起成长，一是对公司的认可度和归属感比较高；第二个是一开始的工资也不需要那么高。很多高工资、高职位的人，在其他地方都会形成一些坏习惯，这些坏习惯要带过来的时候要慢慢地磨、改，还是比较讨厌的。

现在很多公司都参考大公司的职级体系，原因很简单，这就像是一个货币一样，它有流通性。职级起码让大家有一个参考和对标的标准，让你能够就此参考找到最好的一批人，或者找不到也知道自己差在哪儿。这是一种潜规则，其实也是一种明规则。

职级的背后更多的是能力的匹配，所以，如果每个公司都瞄着跟能力匹配去的话，职级这件事情其实还是比较透明的。

- 爱因互动创始人兼 CTO 洪强宁

面试中我会关注候选人的工作内容和工作状态，职级可能会对工作内容有影响，但不会特别关注职级本身。每个面试官都会有自己考察的角度，我本人一般比较在意候选人对新技术的好奇心、对优雅代码架构的追求、发现问题解决问题的敏感度和驱动力。

面试后我们会根据面试官的反馈对候选人的能力进行定级，然后在职级对应的薪资范围内与候选人沟通薪资。不同公司的职级不能进行简单的比较。在进入爱因之后，晋升通道是持续打开的，每半年我们都会做一次人才盘点，根据能力提升情况来确认职级是否需要调整。还是希望大家能够把注意力放在能力提升上。

我觉得建立职级体系有几个好处：

- 可以保持团队待遇整体公平，避免新老倒挂，即新入职员工的待遇大幅超越老员工。有了职级体系，建立能力 - 职级 - 薪资的对应关系，薪资最终是由能力决定的。即使一段时间内由于特殊原因（比如竞争候选人入职，或者面试官判断失误）给出了与能力不匹配的薪资，也可以在未来的定级调整中纠正回来。
- 可以比较直观的反映团队梯队建设情况，给高职级的员工更多权力和责任，持续培养低职级员工成长，有助于团队长久健康发展。

- 员工可以通过职级的提升来了解到自己的成长情况，有意识的去学习和调整自己的工作状态来获得进步。当然这个也可能会带来一定的副作用，就是有可能会让员工变成提职驱动，只挑选那些有助于个人提升职级的事情做，而不一定是对企业有利的事情。管理者可以视公司的发展阶段逐步将职级信息开放。

写在最后

互联网公司的职级，以前我们只能看个热闹，现在我们终于也能看个门道了。其实在技术发展的路线上，慢慢也出现了一个名叫“职业阶梯”的名词。制定职业阶梯的目的是让那些有才华的技术人在职业上有更多的成长和晋升可能性，同时又不需要让他们走管理路线。职业阶梯目前在硅谷已经较为流行，随着互联网技术在中国的持续发展和繁荣，西学东渐，未来的中国技术人肯定也能一直写代码写到 5、60 岁以后。

你在互联网大厂里吗？你现在是一个什么样的职级呢？未来你想成为什么样的技术人呢？快留言告诉大家吧，flag 不立，怎么能够实现呢？

Hadoop 气数已尽：逃离复杂性，拥抱云计算

作者 Matt Asay
译者 杨志昂

虽然大数据依然如日中天，但该领域曾经的领头羊 Cloudera、Hortonworks 和 MapR 三家公司最近步履蹒跚，多少掩盖了其几分风光。作为曾经的数据宠儿，过去筹集到的巨额投资源源不断。例如，英特尔公司就曾向 Cloudera 注入 7.66 亿美元，这还仅仅只是一轮投资的数额！如今，这些大数据领域的重量级公司纷纷被迫瘦身，Cloudera 和 Hortonworks 合并，而 MapR 开始裁员。

与此同时，大数据领域的其他开源供应商（如 Elastic 和 MongoDB 公司）却势头正猛。到底发生了什么事？当然，这背后有种种原因，但其中一个事实是，老牌 Hadoop 供应商把大赌注押在了错误的目标用户上，瞄准的是所谓数据中心的专职架构师。然而，市场已经转向了在云计算环境中寻求自由的个体开发人员。

此消彼长

在那些靠 Hadoop 发家致富的供应商中，MapR 是最新的牺牲品。MapR 公司一度估值超过 10 亿美元，但最近披露的消息是，除非能找到新的投资者，否则公司必须裁员 122 人，这个数量约占员工总数的 25%，而且裁员名单包括其首席执行官 John Schroeder、其他高管以及多名工程师，并且同时准备关闭其总部办公场所。如果真能找到投资者的话，他们必须在 6 月 14 日前签署协议，否则 MapR 的前景将会一片黯淡。

不过，最近大数据领域一直都不太平。根据 LinkedIn 的数据，在过去两年中，MapR 公司已经缩水了 29%。无独有偶，大概是因为 Cloudera 和 Hortonworks 这两家公司无法单独生存，于是它们进行了合并，但在合并之后不久，Cloudera 就宣布了其惨不忍睹的收益，预计收入比分析师预测少了 6900 万到 8900 万美元。与此同时，公司首席执行官 Tom Reilly、联

合创始人兼 CSO Mike Olson 双双宣布辞职。该公司股价随即暴跌 40%。

以上种种结果似乎很容易就被归咎于一个原因：之前的大数据领域被过度炒作，泡沫破灭后回归现实，Hadoop 领头羊已经溃不成军。但这却无法解释为什么大数据领域的其他供应商却依然在蓬勃发展。例如，MongoDB 数据库产品受欢迎程度一直在增长，MongoDB 现在的受欢迎指数大约是 Oracle 和 MySQL 的三分之一，而五年前只有十分之一（https://db-engines.com/en/ranking）。这种受欢迎程度反过来良性地推动 MongoDB 公司的收入增长，最近收入已经跃升了 78%。

同样，Elasticsearch 分布式搜索和分析引擎背后的公司 Elastic 在去年员工数量翻了一番，最近一个季度的收入增长了 70%。许多公司已经转用 Elastic 的产品进行传统的文本搜索和其他更多的搜索，比如英国伦敦的 Stansted 机场就使用 Elastic 工具来追踪和可视化机场内的人员和行李流量，并提供实时分析。

大数据时代的剧本似乎让人看不懂了。像 MongoDB 和 Elasticsearch 这样的技术以及它们背后的公司从来没有被认为能够挑战 Hadoop 和相关产品。然而现在看来，他们确实做到了。为什么会这样？

预报：未来多云

其中一个答案是因为“云”，但它也只是一个多方面综合效应的一个侧面而已。正如 Anaconda 高级副总裁 Mathew Lodge 在一篇文章中所提及的，尽管 Cloudera、Hortonworks 和 MapR 这三家公司都在拼命从现有产品中寻求演进，但 AWS、微软 Azure 和谷歌 Cloud 三巨头打造的一站式云原生服务提供了“完全集成的产品系列，获取成本更低，扩容更便宜”。企业用户的目光纷纷投向了这些服务和产品。虽然 Hadoop 供应商以尽可能快的速度打造自己的云服务，但其速度根本赶不上那些云计算领域的重量级竞争对手。

虽然 Hadoop 在当时是颇具革命性的技术，但与云计算的替代方案相比，它的成本高得离谱。正如 Clint Sharp 所指出的，“Hadoop 的主要应用场景一直是廉价的存储。然而，有了云之后，存储变得更廉价，更何况 S3+EMR 和其他服务的用户体验还提高了千倍不止。”作为传统专有数据仓库的替代品，Hadoop 曾经是很不错的选择，但它现在已经远比不上更现代的技术（甚至是基于云的 Snowflake 数据仓库，等等）。

与此同时，云计算代表着处理数据的新方法。虽然它们本身不是完全同质的替代品，但与 MongoDB 或 Elasticsearch 一样，它们解决了与 Hadoop 相同的问题，而且还简单易用，没有那些令人抓狂的麻烦。正如 MongoDB 的 Joe Drumgoole 所说，“编写有效的分布式 Map-reduce 算法真的非常非常困难。”更糟糕的是，Hadoop 供应商争先恐后地为他们的 Hadoop 产品添加各种开源插件（例如，Impala、Pig、Hive，以及 Flume），还发明了更累赘的“解决方案技术栈”。直到最后，终于有一位观察者这样评价，“没有人知道这些 Hadoop 公司到底在做什么”。

对于一些企业用户来说，或许在这上面费力地付出时间和精力还算值得。然而，对于肩负“把事情做完”任务的个体开发人员来说，他们越来越多倾向于选择更简单直接的替代方案。

使用方便才是王道

Hadoop 及其衍生产品的开箱即用体验确实不忍直视。这与 MongoDB 的用户体验形成鲜明对比。前 MongoDB 高管 Kelly Stirman 认为用户体验是让 MongoDB 在同类产品中脱颖而出的一个关键。这是一种什么体验？一位叫 Tom Barber 的人这样描述：

> 在使用 MongoDB 时，你可以容易地在一台服务器上安装 MongoDB，而不需要在一个糟糕的 VM 上浪费时间。在生产环境中，你可以直接在一台服务器上把它运行起来。你不需要写一大堆代码就可以把它和其他一堆东西连接起来。人人都希望使用这样的数据库…MongoDB 真正做到了很容易让数据流入，也很容易就让数据流出。

TimeScale DB 首席执行官 Ajay Kulkarni 也表示赞同，他补充道：

> 个体开发人员的热爱是 MongoDB 战胜 Hadoop 的原因。MongoDB 聚焦于首次用户体验。而 Hadoop 的运行过程十分繁琐，简直臭名昭著。虽然 Hadoop 供应商针对企业用户提供了一套优秀的销售宣传说辞，但如果没有开发人员的热爱和支持，它的增长就会停滞，市场就会萎缩。

在 MongoDB 和 Elastic 击败 Cloudera 和 MapR 这件事上，虽然把成功因素统统归于开发人员的热爱可能有些夸大其词，但这的确是一个不争的事实。

开发人员 Jake Kaldenbaugh 认为，MongoDB 已经开始“融入”到各种现代应用程序中。随着时间的推移，那些一开始将 MongoDB 应用于并不那么重要的应用程序的开发人员，会将 MongoDB 应用到那些涉及重要业务的应用程序中，而且 MongoDB 还在不断添加新功能（比如多文档事务支持），以支持更复杂的应用场景，但又没有让这些功能变得过于复杂。

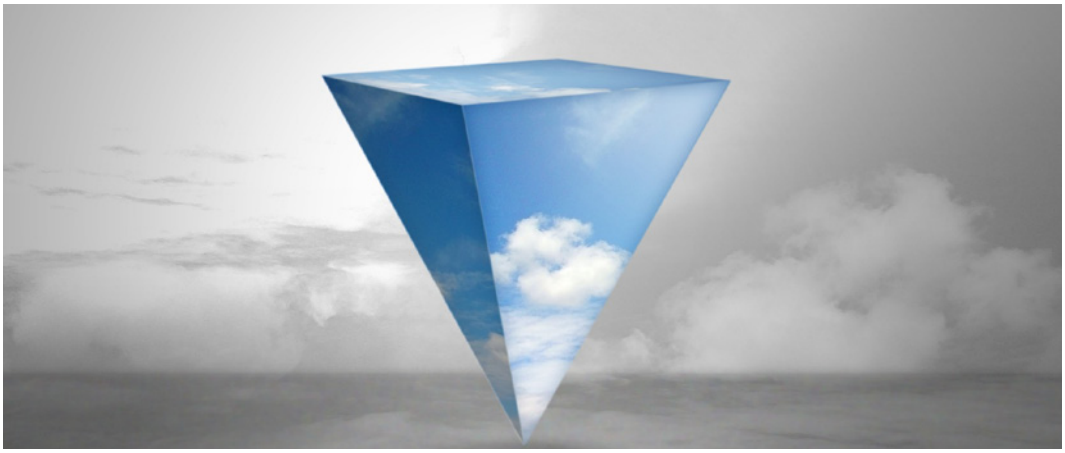
那么，之前的这些大数据巨头公司们将何去何从呢？ Mathew Lodge 已经为他们写下了悼词：

> 在 Cloudera 和 Hortonworks（还有 MapR）作为大数据宇宙中心长达 10 年之后，这个领域的重心已经转移到其他地方。如今领先的云公司并不像 Cloudera 和 Hortonworks 那样运行大型的 Hadoop/Spark 集群，而是在容器基础设施之上运行分布式数据库和应用程序。他们用 Python、R 和其他非 Java 语言进行机器学习。越来越多的企业正转向类似的技术方向，因为它们也希望获得同样的速度和规模效益。那些使用 Hadoop 和 Spark 技术的世界是该紧跟时代做出改变了。

开源数据基础设施的创新日新月异，这既是福，也是祸。创新正在以惊人的速度发生，注定会有一些供应商将在这个飞速发展的过程中破产。

七牛云许式伟：我所理解的架构是什么

作者 许式伟

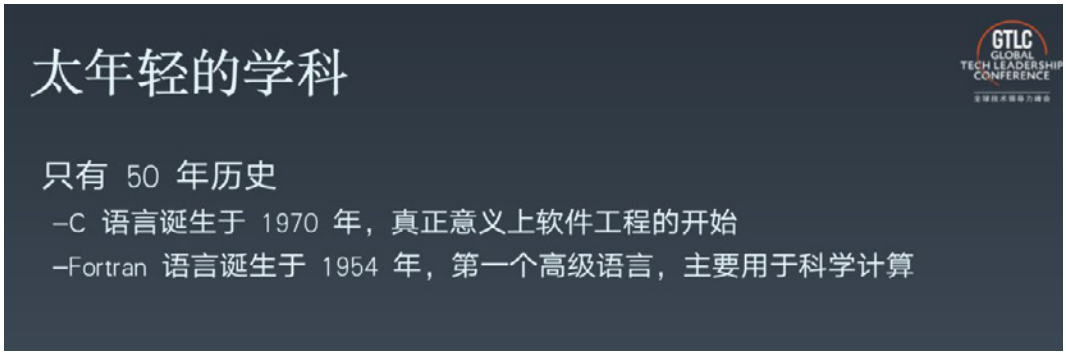


从软件工程说起

大家好！我已经很久没有做技术类的演讲了，因为我最近确实比较忙，很少会出来。为什么会突然又想谈一下架构呢？这是我个人的宿愿，我是技术出身，虽然现在比较少写技术相关的东西，但我在公司内部做了很多分享，分享课里我讲的东西与架构相关的占三分之二，基本都是和架构相关的。

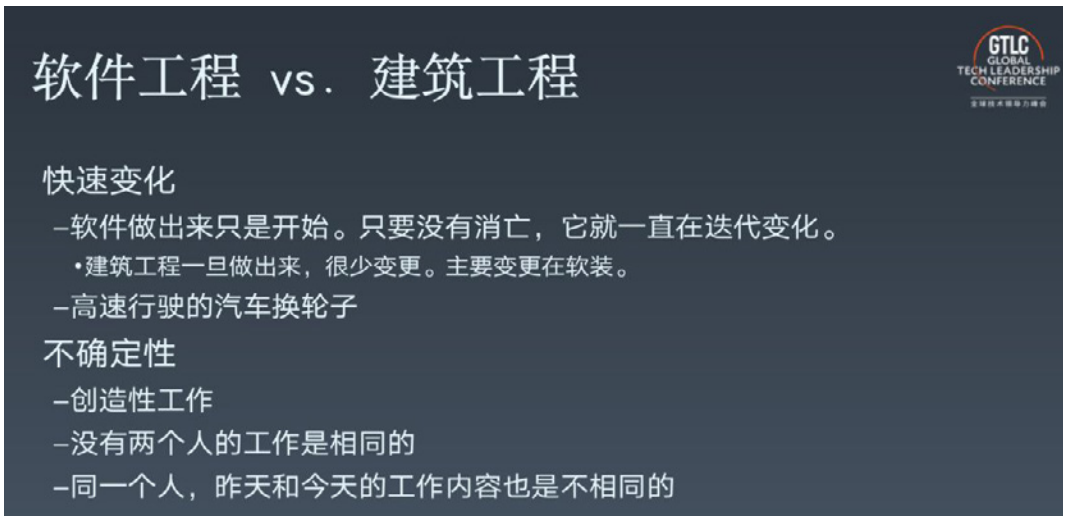
所以今天借这个机会谈一谈我自己理解的架构到底是什么。

国内现在比较少真正意义上符合“架构师”这个词的定位的角色，我们的教育和工作氛围很难出真正意义上的架构师，比较凤毛麟角。我自己理解的架构师是从软件工程概念开始的，也许大家都学过软件工程，但如果我们把软件工程这门课重新看待，这门学科到底谈的是什么？是软件项目管理的方法论？



无论如何，软件工程是一门最年轻的学科，相比其他动辄跨世纪的自然科学而言，软件工程只有 50 年的历史。这门学科的实践太少了，任何一门学科的实践时间短的话，都很难沉淀出真正有创意的实践总结，因为这些经验总结总是需要很多代人共同推动来完成。

为什么只有 50 年时间呢？我们来看看 C 语言，一般意义上可能认为它是现代语言的开始。C 语言诞生于 1970 年，到现在是 49 年。再看 Fortran，它被认定为第一个高级语言，诞生于 1954 年，那时候主要面向的领域是科学计算。Fortran 的程序代码量不大，量不大的时候谈不上工程的概念。这也是为什么软件工程这门学科很年轻，它只有 50 岁，在这样一个年轻的学科里我们对它的认知肯定还是非常肤浅的。



我在极客时间里的课程里一上来就做了软件工程和建筑工程的对比。对比可以发现二者有非常大的区别，具体在于两点：

（1）快速变化。建筑工程在完工以后就结束了，基本上很少会进行变更，除非对它进行软装上的变更，软装更像是今天的软件。但其实软件

工程里，软件生产出来只是开始，而且只要软件的生命周期没有结束，变更就一直存在，很像建筑里的软装一样，而且比软装变化剧烈得多。

（2）不确定性。为什么软件工程有很大的不确定性？因为没有两个人的工作是一样的，虽然大家都在编程，但是编程的内容是不一样的。每个人昨天和今天的工作也是不一样的，没有人会写一模一样的代码，我们总是不停地写新的东西，做新的工作。这些东西是非常不同的，软件工程从事的是创造性的工作。

大家都知道创造是很难的，创造意味着会有大量的试错，因为我们没有做过。这会导致软件工程有非常大的不确定性。

以上这两点都会导致软件工程区别于传统意义上的所有工程，有非常强的管理难度。过去那么多年，工业界有非常多的工程实践，但是所有的工程实践对软件工程来说都是不适用的，因为二者有很大的不一样。

今天站在管理的视角再看软件工程，我们知道管理学谈的是确定性，我们如何去创造确定性是管理学中的追求，否则管理管什么呢？某种意义上来说管理学的目的就是要抑制不确定性，产生确定性。比如说开发的工期，时间成本是否能确定。其次，人力成本，研发成本和后期运维的成本是不是确定性的。所以软件项目的管理又期望达到确定性。这是一对矛盾。软件工程本身是快速变化的，是不确定的。但是软件工程管理又希望得到确定性，这就是软件工程管理上的矛盾。我们的目标是在大量的不确定性中找到确定性，这是我认为这件事情最核心的点。

程序员的三个层次

软件工程管理到底在管什么？和所有的管理活动一样无非就是人和事。所有的工程项目都希望找到最好的人，当然是在能给出的预算以内找到最好的人，有的人可能找不起。不同项目最大的差别就是事，不同的事在哪里？从做事的角度来讲我们招到的人可能会分三个层次（程序员三个级别），大家经常开玩笑说我是做搬砖的，所以第一个 level 我把他叫软件搬砖师，再然后是软件工程师、软件架构师。

软件搬砖师可以有很多。但今天数量其实还不算太多，因为我们知道这门学科只有 50 年的历史。但是好的一点是，产生软件搬砖师并不难，我做了一个长达四年的实践：从小学二年级开始教小学生编程。结论是做搬砖师不难，小学生也能做到。这是很有意思的一件事情，编程并不是非

常复杂的学问，只要具备基本的逻辑能力，把常规的业务代码按部就班地垒出来，基本上可以算打到搬砖师水准。我自己认为这并不难。

软件工程师会相对难一些，我心目中的软件工程师首先在代码上会非常追求可读性、可维护性。另外，毕竟我们工程是群体协作，所以在群体协作上还是有自己的方法论和思考。比如说代码评审、单元测试。在我看来搬砖师和工程师的区别有很大不同。只要看他写的代码有没有注意可维护性，会和同伴交流的时候刻意去追求让同伴更好地理解自己的思想，是不是对单元测试比较抗拒，是不是比较乐意去做代码评审并且非常认同这件事情的价值，基本上通过这些事情就可以评判这个人是搬砖师还是工程师。

软件架构师的能力要求

谈到软件架构师，由于我毕业后两年在从事架构性质的工作，因此对软件架构师的特性有一些总结。首先在用户需求上，有判断能力和预见能力，此处的判断可以理解为对需求的鉴别，虽然这可能与产品经理最为相关，但架构师需要具备自己的判断力，当然这也包括对未来需求的预见能力；产品迭代上，有规划能力，判断需求哪些应该先满足，哪些后满足。架构师应该源于程序员，但不应局限于程序员视角。系统设计上，有分解和组合能力。技术选型上，有决策力。技术选型应该被认为是架构的一部分，我们非常反对开发人员随意选用开源组件，这是一件需要认真探讨的事情。人力资源上，有统筹能力，通俗地讲是“看菜做饭（看人下菜）”。

综上不难看出，架构师对综合能力要求比较高。这是因为我认为架构师需要对软件工程的结果负责，在不确定性和快速变化中寻找确定性。全局看软件发布流程，其比较重要的子过程有：需求分析（需求梳理 => 产品定义），系统设计（子系统划分 => 模块定义），模块设计（模块详细设计），编码实现，单元测试，代码评审，集成测试，灰度发布，正式发

布等一系列过程。虽然有些过程看起来不属于架构师的范畴，但是这些活动过程属于软件工程的一部分，架构师一样需要全面参与把控。如果没有架构师把控就没有人观察得到全貌。正因为如此，软件架构师的要求相对较高。

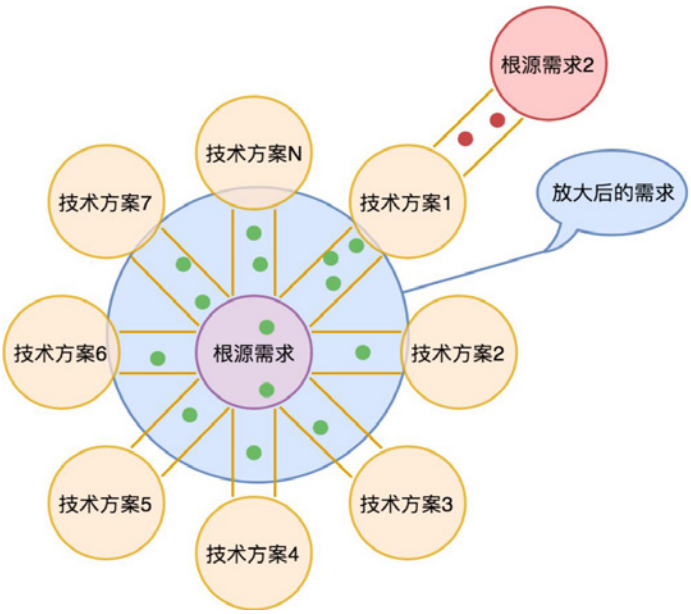
如上所言，软件架构师需要具备产品经理的部分能力，因为需要对用户需求进行分析，并进行判断和预判，以及对产品迭代优先级进行把控。我自己习惯用如下图片表达软件架构师和产品经理之间的关系。

我认为，产品是“桥”，连接了两端，分别是用户需求和先进的技术。我一直认为，用户需求的变化非常缓慢，那么为什么产品会产生迭代？这是因为技术在迭代。本质上讲，产品迭代是技术迭代导致的需求满足方式的变化，所以产品实际上是一种需求满足的方式。

从这个意义上讲，架构师更多是从技术方案的角度看产品，而产品经理更多是从用户需求来看，但二者一定会碰头，只要能力提升到角色所期望的样子，越厉害就越具备两侧的能力。所以我认为，产品经理和架构师是一体两面，本质上对人的能力、诉求是相通的。产品经理在做产品架构，架构师在做技术架构，但最终目的一样。

从产品和需求视角看架构师

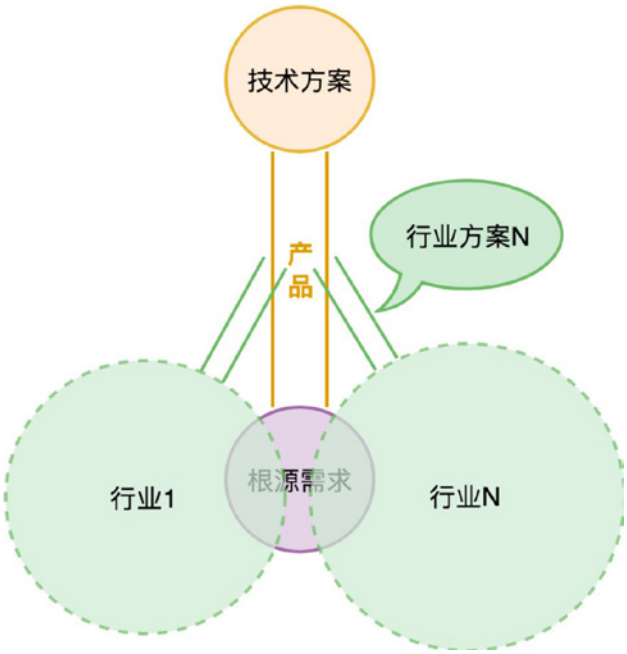
如果展开讲解产品定义过程，首先需要进行需求梳理，关心用户反馈。但是，很多用户反馈并不代表其根本性需求。有很多用户反馈需求的时候，往往已经带着他自己给出的解决方案。这种需求反馈已经属于二次加工的需求，而非原始需求。这个时候我们要多问多推敲，把它还原到不带任何技术



实现假设的根源需求。

如上图所示，根源需求可能会有非常非常多的技术方案可以满足它。我们下面示意图中的小圆点是一个个用户反馈的需求。在用户提这些需求的时候，往往可能会带着他熟悉的技术方案的烙印。

产品都是通过提供相应的技术方案在满足用户的根源诉求，但技术一直在迭代进步，从而导致原有的解决方案过时落后，这种情况下需要新的解决方案出现。如果对用户反馈的需求全部满足，产品就会变得十分庞大，编程一个四不像的东西。



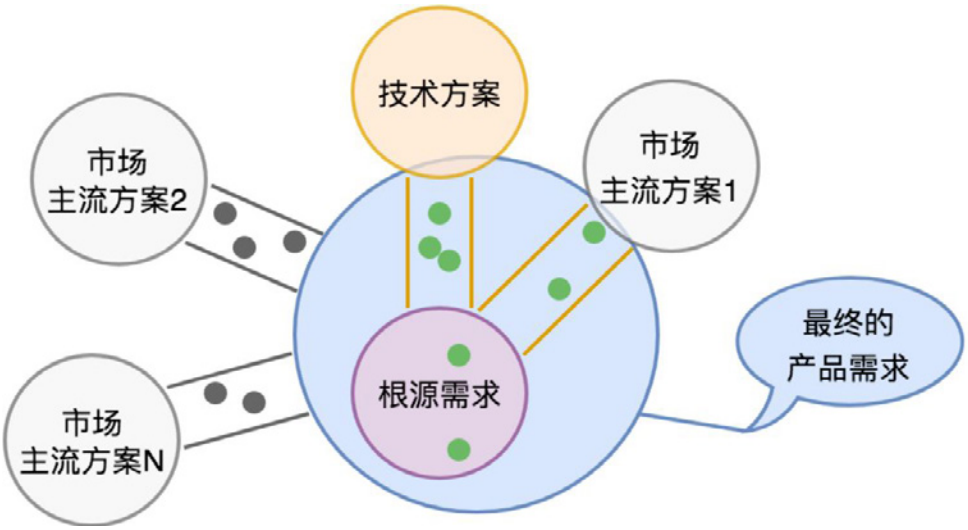
其次，在这个过程中，有些用户需求是稳定的，有些是变化的。举例来说，计算机系统结构从计算机诞生之后到现在没变过，但电子设备的形态发生了很大变化，从最早的大型机，到个人电脑，到笔记本，到手机，再到手表，形态变化剧烈。但为什么计算机系统结构能够适应需求而不用改变架构，这其实是非常值得思考的事情，其根源就是对变化点的抽象，找到系统需求的变化点，预见变化并做对应的开放式设计。本质上讲，架构师关心产品的核心根源就是预测变化。

最后，理清产品边界。同样以计算机为例，经过多轮迭代，多样化外设（键盘等）变化较大，但 CPU、内存演进较小，所以在变化点上做相应的开放式设计是必要的。同样的，需要与合作伙伴做边界设定，把变化

开放出去让合作伙伴做，只有这样的产品才能达到较好效果。

从产品和解决方案角度来看，产品往往需要适应很多行业，但这个过程会让产品变得非常庞大。在我看来，产品应该为行业解决方案提供能力，行业解决方案优先选择合作伙伴做，以更加开放的心态看待这件事情，避免把行业方案视作产品的一部分。

梳理需求中比较关键的点是市场策略，需要解决的需求有非常多现成的方案，但哪些方案是主流的，哪些是最关键的都需要思考。虽然不能放大产品需求覆盖面，但也需要为某些关心既有市场的玩家做桥梁，这些桥梁也是产品的功能点。我倾向于认为关键市场可能会把既有玩家的能力适配到产品上作为很重要的功能，但是大部分市场主流方案我们还是提供桥，而不是自己解决掉。



从技术视角看架构师

以上是从产品和需求维度看架构师，从技术视角看，架构师很重要的能力是具备技术的全局视角，所谓的技术全貌是指从底到上的核心骨架，比如最底下的硬件结构、操作系统、编程语言，甚至浏览器等，只有掌握每一层的核心思想，才能在架构设计中没有技术盲点。

从培养架构师的角度来看，为什么真正意义上的架构师比较难找？这是因为需要构建两个层次的能力：

1. 懂用户、懂市场，有一定市场洞察能力。作为技术人员，可能会不自觉、甚至不愿意和用户打交道，更希望坐在家里安静码代码。但是，作为架构师，不和用户打交道，成长会比较受限，不接触用户就无法理解用

户需求，亲自和用户打交道倾听来的需求和探讨完全不一样。因此，架构师要尊重用户反馈，并学会思考需求分析和推演，这比技术能力更重要。架构的第一步就是需求分析，如果需求分析没做好，后续自然没办法做得很极致。

2. 建立技术上的全局视角。

以上两点是架构师最核心的两个能力。

Kubernetes 已足够成熟？详细解读 1.15 新版本的多项关键特性

作者 华为云原生团队

2019 年 6 月 20 日，Kubernetes 重磅发布了 1.15 版本，不管你是 Kubernetes 用户，还是 IT 从业者都不能错过这个版本。

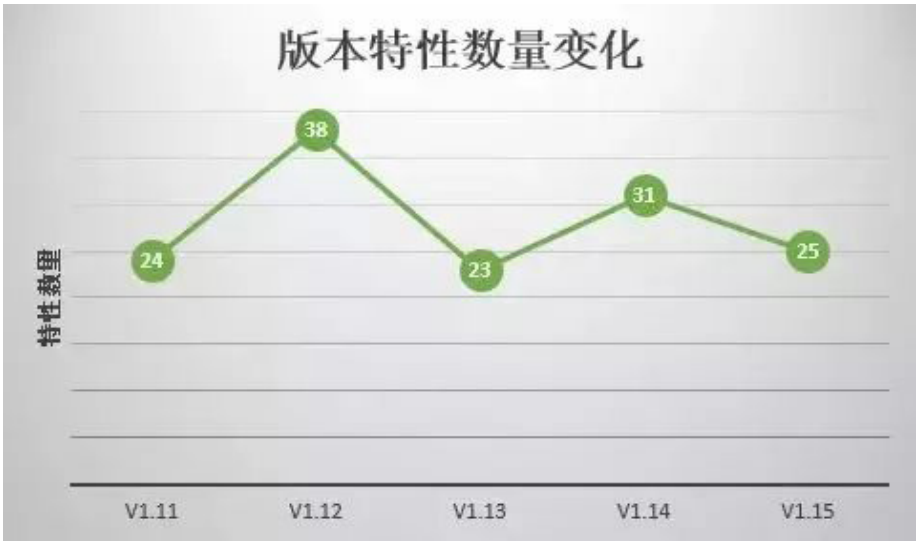
1.15 版本主要围绕可扩展性展开，北向 API 接口方面 API Machinery SIG 致力于催熟 CRD 以提升 API 可扩展性，南向插件集成方面，Storage SIG 和 Node SIG 则分别对 CSI 和设备监控插件可扩展性进行了优化。另外 Kubeadm 对 HA 集群配置也达到 Beta 可用，并发优化了证书管理相关功能。

本文我们先从宏观上了解一下近期版本的变化趋势，然后再开始 1.15 版本的重点特性解读。

Kubernetes 持续热情高涨

在展开解读 1.15 的关键新特性之前，让我们先来回顾下社区过去几个版本的特性发布情况。值得一提的是：无论从新特性数量，还是特性的成熟度分布来看，Kubernetes 依旧保持着相当的活力，社区开发者用实际行动回应了业界盛传“Kubernetes 已经足够成熟，项目正在变得无聊 (Boring)”的说法。

各版本新特性数量基本持平



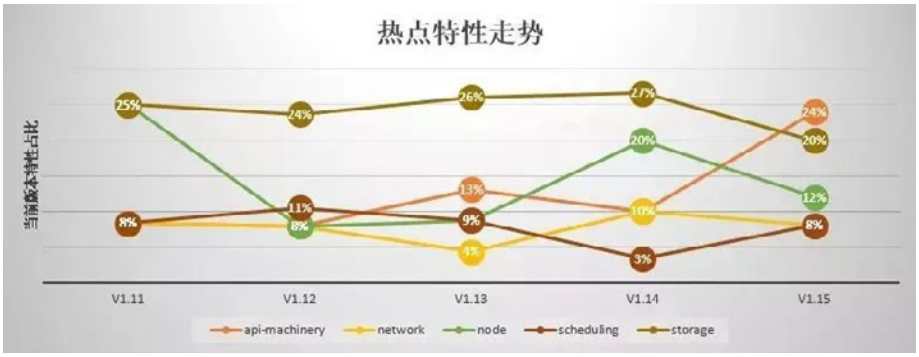
从新特性数量上看，Kubernetes 过去一个版本发布的特性数量并无明显趋势性变化。由于特性颗粒度的差异，每个版本发布的新特性数量存在小幅波动，属于正常现象。

看特性成熟度分布，Alpha 特性比例依旧可观

对比分析过去几个版本的特性成熟度，不难发现 Alpha 新特性的占比稳定在 20%~40% 之间。按照社区惯例，当一项全新的功能被添加时，会在特性说明中打上 Alpha 标记。持续稳定甚至小幅上涨的 Alpha 特性占比说明社区仍然在大量开发全新的特性，而不是满足于既有功能的加固。

Kubernetes 重点特性解读

经过分析 Kubernetes 近几个版本的变化，我们发现特性主要集中在 Storage、Node、API-Machinery、Network、Scheduling 等 SIG，而这些 SIG 大都致力于可护展性增强，以便于下游用户在享受稳定核心的同时，轻松扩展定制自己的插件。



从最新发布的 1.15 版本来看，以下几个变化需要重点关注。

API 聚焦可扩展性增强

Kubernetes API 的扩展能力主要由 CRD（Custom Resource Definition）以及 Admission Webhook 提供。1.15 版本在这两方面都有重要的更新。

CRD，大步迈向 GA

CRD 的新开发主题一直都围绕着数据一致性以及提供更加接近 Kubernetes 原生 API 的能力。用户不应该感知到到底是以 CR（Custom Resource）还是以原生的资源对象形式与 Kube APIServer 进行交互。社区目前正迈着大步，计划将在接下来的某个版本中将 CRD 以及 Admission Webhook GA（升级为稳定版本）。

朝着这个方向，社区重新思考了基于 OpenAPI 的 CRD 验证模式，从 1.15 开始，根据结构模式（Structural Schema）的限制检查每个字段。这基本保证了能够像原生的 API 对象一样提供完整的 CR 校验能力。在 v1beta1 API 中，非结构模式（non-Structural Schema）仍然保持工作状态。但是任何严格的 CRD 应用程序都应该在可预见的将来迁移到结构模式。

1. CRD 多版本之间通过 Webhook 转换特性 [Beta]

从 1.15 版本开始，支持运行时不同版本之间的转换，长期目标是让

用户像原生 API 资源一样使用 CRD 的多版本。这一特性是 CRD 在 GA 道路上一步重大的飞跃。

2. CRD OpenAPI 发布特性 [Beta]

CRD 的 OpenAPI 发布特性将会在 Kubernetes 1.15 中 Beta，当然只是针对结构模式的 CRD。次特性对于用户自定义 API，提供了与 Kubernetes 原生 API 一样的文档说明。

3. CRD 未知字段裁剪（Prune）[Beta]

CRD 未知字段裁剪特性是针对发送到 Kube APIServer 的对象中未知的字段进行移除，并且不会持久化存储。用户可以通过在 CRD 对象设置 spec.preserveUnknownFields: false 使用此未知字段裁剪特性。此特性对于数据一致性及安全都有一定的帮助。

4. CRD 默认值设置 [Alpha]

Kubernetes 1.15 结构模式的 CRD 默认值设置特性作为 Alpha 特性可用。用户可以通过 OpenAPI 校验模式的 default 关键词指定默认值。避免了用户原来只能通过 Admission Webhook 方式设置 CRD 对象的默认值带来的额外开发消耗。

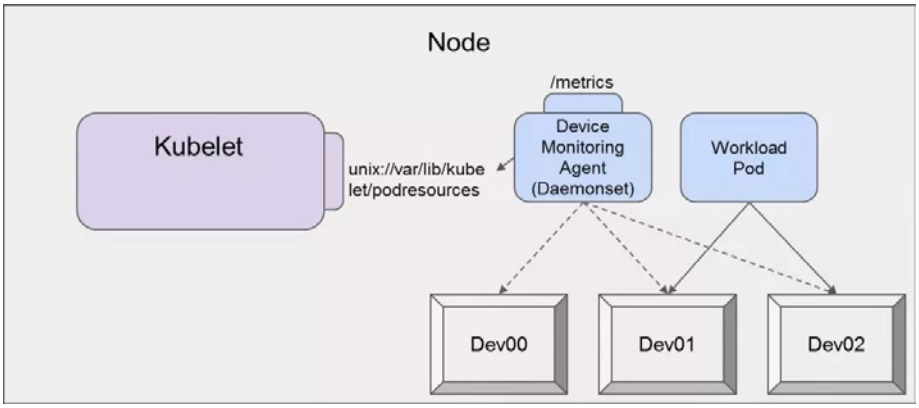
Admission Webhook 重新调用（Reinvocation）与增强

1. Mutating 和 Validating Admission Webhook 已经成为扩展 API 的主流选择。在 1.15 以前，所有的 webhook 只会按照字母表顺序调用一次，这样就会导致一个问题一个更早的 webhook 不能应对后面的 webhook 的更新，这可能会导致未知的问题，例如前面的 webhook 设置某个 pod 的启动参数，而随后的 webhook 将其更改或者移除了。1.15 版本引入 Reinvocation 特性允许用户通过 MutatingWebhook 配置 reinvocationPolicy: IfNeeded 指定 Mutating Webhook 重新调用。
2. Admission Webhook 引入了 objectSelector 来控制通过标签选择特定的对象进行准入控制。
3. 允许 Admission Webhook Server 使用任意的端口（不限制只能是 443）。

Node SIG 提供监控插件框架用于异构硬件监控

新版本很好的解决了异构硬件设备监控难题，现在不需要修改 Kubelet 代码就可以实现各种指标（如 GPU、显存利用率等）监控。

新版本通过新的监控插件框架将 Kubelet 与设备监控处理逻辑解耦，很好的解决了以往 Kubelet 代码管理和 K8s 集群运维之间的痛点。



由图可见本框架主要包含 Kubelet 和 Device Monitoring Agent 两大部分：

1、Kubelet

Kubelet 增加了一个 GRPC 服务，监听在 /var/lib/kubelet/pod-resources/kubelet.sock，提供 pod resources 的 list 查询接口。

2、Device Monitoring Agent

- Device Monitoring Agent 提供 metric 接口对外提供设备监控指标查询功能。
- Device Monitoring Agent 向 Kubelet 的上述 GRPC 服务发送 List 请求，取得所有 pod resources(包含节点上所有 pod 的所有 container 分配的 device 类型和 device id)。
- Device Monitoring Agent 根据设备类型过滤获取容器的设备 ID，通过设备驱动接口获取容器使用的设备的监控指标，并把二者关联到 container 、pod 上。

新的框架将会带来诸多便利：

1. 设备监控功能与 Kubelet 接口解耦

a. Kubelet 不需要提供设备的监控指标；

b. 设备的新增监控指标不需要升级 Kubelet，而是更新设备监控代理版本即可；

c. 新增设备不需要升级 Kubelet，而是增加部署设备监控代理的 DaemonSet 即可；

d. 未解耦前，Kubelet 会检测所有支持的设备是否存在，即使节点并没有安装该设备。

2. 设备供应商负责发布和维护设备监控代理，设备供应商在如何运行和监控它们方面拥有深厚的专业知识，可以提供权威的设备监控代理。

3. K8s 集群的管理员只需要安装需要的设备监控代理即可。

Scheduling SIG 发布 Scheduler Framework 加强调度器扩展定制能力

Scheduler Framework 终于在 1.15 迎来了 Alpha 版本。Scheduler Framework 最早在 2018 年初提出，主要是为了解决日益增加的定制化调度需求。Scheduler Framework 在原有的 Priority/Predicates 接口的基础上增加了 reserve, pre-bind 等十几个接口用于相应前处理及后处理。在 1.15 中，Scheduler Framework 实现 QueueSort, Prebind, Postbind, Reserve, Unreserve 和 Permit 接口，这些接口为后继更多的调度策略提供了基础，比如 gang-scheduling。

然而，出于设计上的延续性考虑，目前 Scheduler Framework 仍以 Pod 为单位进行调度，而计算类（离线）任务调度往往需要考虑工作负载中相关的 Pod 按组进行调度，例如 faire-share。现在的 Scheduler Framework 还不能有效的解决。所幸针对这一能力缺失，社区已经有相应的项目在 Kubernetes 上支持计算类（离线）任务，例如 Volcano (http://github.com/volcano-sh/volcano)。

Storage SIG 持续改善 CSI

在 Kubernetes v1.15 中，SIG Storage 继续工作，以支持将 in-tree 插件迁移到 CSI（Container Storage Interface，容器存储接口）。SIG Storage 致力于使 CSI 具有与 in-tree 功能相同的特性，包括调整大小、内联卷等功能。SIG Storage 在 CSI 中引入了一些新的 Alpha 功能，这些功能在 Kubernetes 存储子系统中还不存在，如卷克隆（volume cloning）等。

卷克隆允许用户在提供新卷时将另一个 PVC 指定为“数据源（Data Source）”。如果底层存储系统支持此功能并在其 CSI 驱动程序中实现“CLONE_VOLUME”功能，则新卷将成为源卷的克隆。

Kubeadm 的 HA 集群配置达到 Beta 可用

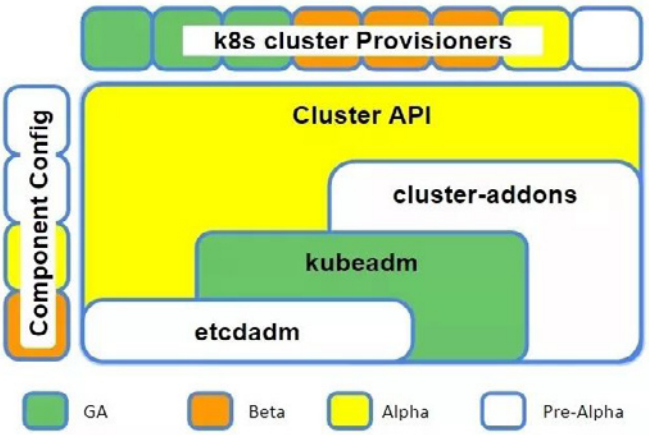
作为社区内置安装工具的 Kubeadm，何时支持部署 HA 集群一直是个热门话题。在本次发布的 1.15 版本中，Kubeadm 对 HA 集群配置的支持终于达到 beta——用户可以直接使用 init 和 join 两个 Kubeadm 命令来部署 HA 集群。乐于折腾的用户可以参考社区文档在小范围生产环境中使用。

Kubeadm 的证书管理功能在 1.15 中也得到了改进，用户可以在证书失效前通过 kubeadm alpha certs renew 平滑地刷新它们。

Kubeadm 的 configuration(API) 在 1.15 版本中引入了 v1beta2 版本，主要是增加了一组证书加解密密钥的配置字段 CertificateKey，便于用户在使用 Kubeadm 部署 HA 控制面时，直接使用被加密保存在集群 Secret 中的证书。此外，用户可以通过 kubeadm alpha certs certificate-key 命令来直接生成这对密钥。

关于跟多 Kubeadm 当前相关的 Alpha 特性，可以查看相关官方文档。

随着 Kubernetes 越来越多地被用在多云、混合云部署场景，集群生命周期管理的标准化一直是众人期待的一大方向。自 v1.13 版本 GA 之后，Kubeadm 的改动主要聚焦在优化使用体验上。而集群生命周期整体流程的打通，特别是实现跨平台的集群管理，配合日渐活跃的 ComponentConfig 以及 Cluster API 两个新项目，相信很快会有实质性的进展。



附图：集群生命周期 SIG 相关项目成熟度

小结

经过 5 年的发展，Kubernetes 核心的基本功能已相对稳定，具备大规模生产可用水平。但是客户需求往往是多种多样的，社区从很早就意识到这个问题，因而提供了各种接口（CRD、CRI、CSI、CNI 等），希望在保持 Kubernetes 独立的条件下，尽量满足用户差异化的需求，这从 1.15 发布的主要特性也可以看出来。

华为云原生团队坚定的认为，Kubernetes 社区未来会继续着重围绕可扩展性、易用性以及稳定性规划发展路标，丰富平台面向各种应用场景的功能接口，以稳定、健壮、高可扩展的平台推动云原生应用生态百花齐放。

利用 CSI 和 Kubernetes 实现动态扩容

作者 华为云原生团队

Kubernetes 本身具有包含了具有大量用例且功能强大的存储子系统。然而，如果我们利用 Kubernetes 建设关系数据库平台，就需要面临一个挑战：建立数据存储。本文用来讲述如何扩展 CSI(容器存储接口)0.2.0 同时整合 Kubernetes，并且展示了动态扩容的重要性。

简介

容器编排技术具有很大的发展空间。开发者们希望能通过开源解决方案来重新设计运行在虚拟化基础设施和裸金属上的独立应用程序。

对于可扩展性和技术成熟度两方面，Kubernetes 和 Docker 处于业内的顶端。关系数据库对于迁移至关重要，但是将独立应用程序迁移到像 Kubernetes 这样的分布式编配十分具有挑战性。

对于关系型数据库来说，我们应该关注其存储功能。Kubernetes 本身具有强大的存储子系统，其功能强大，包含用例广泛。如果我们利用 kubernetes 建设关系数据库平台，就需要面临一个挑战：建立数据存储。但是 Kubernetes 还有一些基本的功能没有实现，尤其是动态扩容。这听起来很无聊，但是创建、删除、挂载和卸载等操作非常必要。

目前，扩容只能与存储提供程序一起使用，例如：

- gcePersistentDisk
- awsElasticBlockStore
- OpenStack Cinder
- glusterfs
- rbd

为了启用这个特性，我们需要设置 feature-gate expandpersistentvolume 为 True，并打开 PersistentVolumeClaimResize 准入插件。一旦启用了 PersistentVolumeClaimResize，调整存储大小的功能将被 allowVolumeExpansion 设置为 True 的存储类开启。

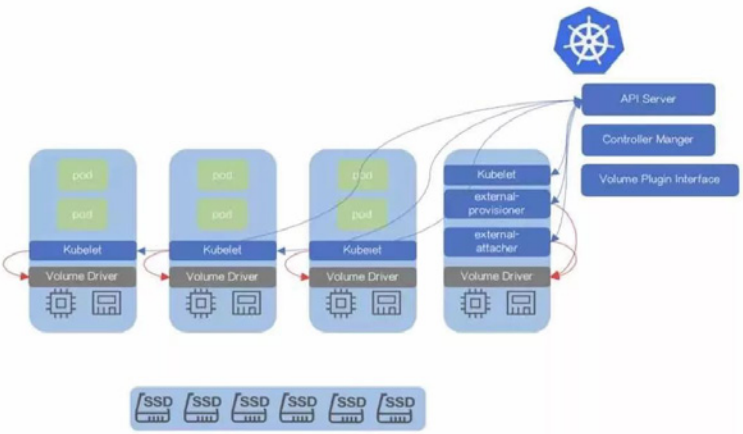
不幸的是，即使底层存储提供程序具有此功能，通过容器存储接口 (CSI) 和 Kubernetes 动态扩容依然不可用。

本文将给出 CSI 的简化视图，然后介绍如何在现有的 CSI 和 Kubernetes 上引入一个新的扩展卷特性。最后，本文将演示如何动态地扩容。

容器存储接口（CSI）

为了更好地理解我们之后的工作，首先我们需要知道容器存储接口是什么。目前来看，Kubernetes 现有的存储子系统依然存在很多问题。存储驱动程序代码维护在 Kubernetes core 存储库中，这一问题使测试十分不便。但除此之外，Kubernetes 还需要授权存储供应商将代码签入 Kubernetes 核心存储库。但是理想情况下，这种需求应该在外部实现。CSI 的设计目的是定义一个行业标准，该标准将使支持 CSI 的容器编排系统可用的存储提供者能够使用 CSI。

这张图描绘了一种与 CSI 结合的高级 Kubernetes 原型。



- 引入了三个新的外部组件来解耦 Kubernetes 和存储提供程序逻辑
- 蓝色箭头表示对 API 服务器调用的常规方法
- 红色箭头表示 gRPC 调用卷驱动程序

详见文档。

扩展 CSI 和 Kubernetes

为了实现在 Kubernetes 上扩展卷的功能，我们应该扩展多个组件，包括 CSI 规范、“In-Tree”卷插件、外部供应器和外部连接器。

扩展 CSI 规范

最新的 CSI 0.2.0 中还没有定义扩展量的特性。应该引入新的 3 个 rpc，包括 RequiresFSResize，ControllerResizeVolume 和 NodeResizeVolume。

```
service Controller {
rpc CreateVolume (CreateVolumeRequest)
  returns (CreateVolumeResponse) {}
……
rpc RequiresFSResize (RequiresFSResizeRequest)
  returns (RequiresFSResizeResponse) {}
rpc ControllerResizeVolume (ControllerResizeVolumeRequest)
  returns (ControllerResizeVolumeResponse) {}
}
service Node {
rpc NodeStageVolume (NodeStageVolumeRequest)
  returns (NodeStageVolumeResponse) {}
……
rpc NodeResizeVolume (NodeResizeVolumeRequest)
  returns (NodeResizeVolumeResponse) {}
}
```

扩展“In-Tree”卷插件

为了扩展 CSI 规范，csiPlugin 接口需要在 Kubernetes 上实现。csiPlugin 接口将会扩展 PersistentVolumeClaim 用来代理 ExpanderController。

```
type ExpandableVolumePlugin interface {
VolumePlugin
ExpandVolumeDevice(spec Spec, newSize resource.Quantity,
```

```
oldSizeresource.Quantity) (resource.Quantity, error)
RequiresFSResize() bool
}
```

实现盘卷驱动

最后，为了抽象实现的复杂性，我们应该将单独的存储提供程序管理逻辑硬编码到 CSI 规范中定义的以下函数中：

- CreateVolume
- DeleteVolume
- ControllerPublishVolume
- ControllerUnpublishVolume
- ValidateVolumeCapabilities
- ListVolumes
- GetCapacity
- ControllerGetCapabilities
- RequiresFSResize
- ControllerResizeVolume

示例

让我们用一个具体的用户案例来演示这个特性。

- 为 CSI 存储供应器创建存储类

```
allowVolumeExpansion: true
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: csi-qcfs
parameters:
 csiProvisionerSecretName: orain-test
 csiProvisionerSecretNamespace: default
provisioner: csi-qcfsplugin
reclaimPolicy: Delete
```

```
volumeBindingMode: Immediate
```

- 跨 Kubernetes 集群部署 CSI 卷驱动程序，包括存储供应器 CSI -qcfsplugin
- 创建将由存储类 csi-qcfs 动态提供的 PVC qcfs-pvc

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: qcfs-pvc
 namespace: default
....
spec:
 accessModes:
 - ReadWriteOnce
 resources:
   requests:
     storage: 300Gi
 storageClassName: csi-qcfs
```

- 创建 MySQL 5.7 实例来使用 PVC qcfs-pvc
- 为了反映完全相同的生产级场景，这里分为两种不同类型的工作负载，包括 :
  1. 批量插入，使 MySQL 消耗更多的文件系统容量
  2. 增加查询请求
- 通过编辑 pvc qcfs-pvc 配置动态扩展容量

Prometheus 和 Grafana 的集成使我们可视化相应的关键指标。

我们注意到中间的读数显示 MySQL 数据文件大小在批量插入时缓慢增加。同时，底部读数显示文件系统在大约 20 分钟内扩展了两次，从 300G 扩展到 400G，然后是 500G。同时，从上面的读数可以看出，整个扩容过程立即完成，对 MySQL QPS 影响不大。

结论

无论基础设施应用程序运行在何处，数据库始终是一个关键资源。用高级的存储子系统来完全支持数据库需求非常重要。这将有助于推动更广泛地采用云本地技术。

扫码关注InfoQ公众号

Geekbang. 极客邦科技 | InfoQ