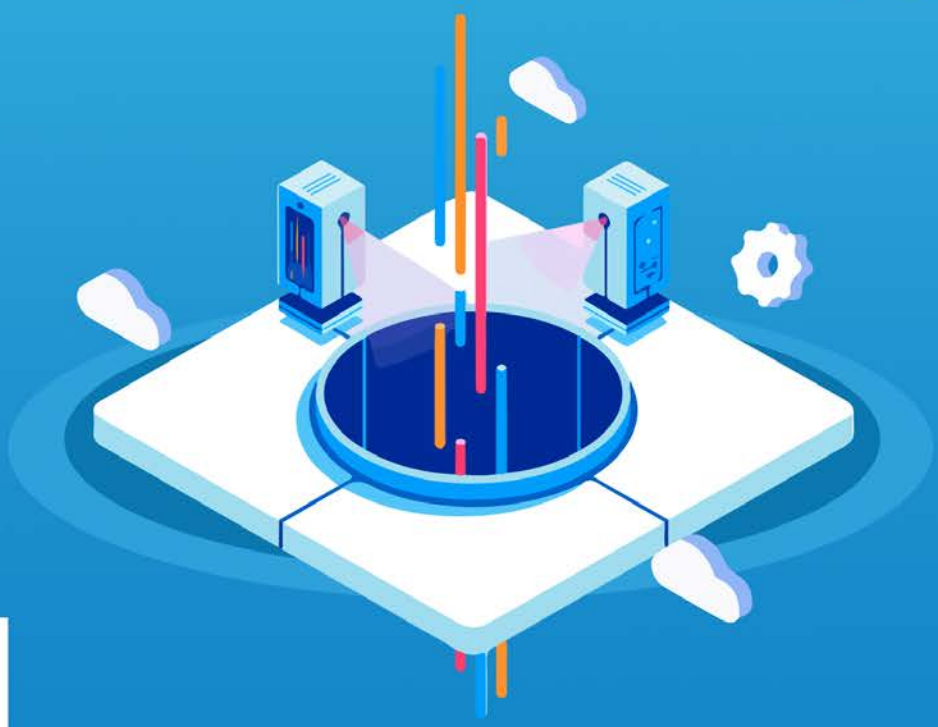
架构师

ARCHITECT

1月刊

热点

2018年终盘点：我们处在一个什么样的技术浪潮当中？



Geekbang. 极客邦科技 | InfoQ

CONTENTS / 目录

热点 | Hot

阿里重磅开源 Blink：为什么我们等了这么久?

理论派 | Theory

被高估的 2018：深度学习发展并没有想象的快

推荐文章 | Article

去哪儿网消息队列设计与实现

观点 | Opinion

2018 年终盘点：我们处在一个什么样的技术浪潮当中?

专题 | Topic

2019 年 Vue 学习路线图

特别专栏 | Column

全球 6 大数据中心，日均 10 亿日志场景下的高可用实践

UCloud 大规模数据中心网络管理系统建设之路

架构师

2019 年 1 月刊

本期主编 徐川

流程编辑 丁晓昀

发行人 霍泰稳

提供反馈 feedback@geekbang.com

商务合作 hezuo@geekbang.org

内容合作 editors@geekbang.com

卷首语

开源路在何方?

作者 徐川

最近，包括 MongoDB 在内，多个知名开源项目纷纷修改开源协议，限制商业化使用，这个举动是对云计算厂商利用开源项目而没有任何回馈的反击。但是，这种做法很难说有多少效果。

笔者和一些开源界人士交流，不少人认为基于开源项目做咨询和技术支持服务的这种商业模式已经快走到尽头。

因为这些单个的开源项目需要组合到一起才能使应用运行起来，而云厂商已经把这些技术都集成到云计算资源里，并且提供技术支持，用户没有理由再为单个技术而付费了，这个趋势并不会因为修改开源协议而改变。

有人说，基金会模式是这些开源项目的出路，比如加入到 CNCF、Apache 基金会，大家一起抱团取暖，云厂商每年交一些会费成为基金会的会员，用这些钱来维护开源项目和做一些布道活动。同时基金会也有更大的力量和云厂商进行谈判，以保护开源项目背后的公司或团队。

这种模式也许可行，但是，基金会能容纳的开源项目毕竟有限，而且基金会如果扩张过快，会有治理方面的隐忧，基金会也会对开源公司有所限制，不利于商业化的探索。

还有人悲观的说，开源项目做大之后，被收购是唯一的出路。他们的证据就是连开源商业化的代表公司，红帽都被收购了。但这样的话，处于竞争状态的云厂商不会使用对方的技术，不同的云计算公司将采用不同的技术，开源项目不再成为事实标准，这让开源项目本身的价值大减，用户不会再去专门学习一个平台绑定的技术了。

开源运动至今已有近 30 年，方便的获取和宽松的协议促进了技术的分享传播，也间接推动的技术的飞速发展，过去开源精神和开源的规则得到了比较好的贯彻，但当初制定这些规则的人肯定没有料想到今天会出现巨无霸型的云计算公司，现在，是时候重新思考开源的未来了

主办方 Geekbang. 极客邦科技 InfoQ

大咖助阵

联席主席：程立（鲁肃）
蚂蚁金服 / 首席技术官

联席主席：于旸(TK)
腾讯 / 玄武实验室总监

联席主席：祁安龙
百度搜索公司
首席架构师 / 技术委员会主席

联席主席：洪强宁
爱因互动 / 创始人兼CTO

专题出品人：程劭非（寒冬）
自由职业 / 前端工程师

专题出品人：董志强（Killer）
腾讯 / 云鼎实验室
云基础安全中心负责人

分享嘉宾

Stuart Douglas
Red Hat
Undertow project leader

Boris Scholl
Microsoft
Azure Compute Product Architect

Emily Jiang（蒋丰慧）
IBM MicroProfile CDI首席架构师

王明刚
英特尔
软件工程师

李鹏
阿里云
容器服务资深架构师

姜承尧
腾讯金融科技
副总监

赵健博
快手 高级架构师
大数据架构团队负责人

姜宝琦
百度
资深工程师

优惠

100+技术大咖 实战解析

如果您有任何需要或问题，请联系我们：

购票热线：010-53935761 票务微信：qcon-0410

阿里重磅开源 Blink：为什么我们等了这么久？

作者蔡芳芳

今年，实时流计算技术开始步入主流，各大厂都在不遗余力地试用新的流计算框架，实时流计算引擎和 API 诸如 Spark Streaming、Kafk Streaming、Beam 和 Flink 持续火爆。阿里巴巴自 2015 年开始改进 Flink，并创建了内部分支 Blink，目前服务于阿里集团内部搜索、推荐、广告和蚂蚁等大量核心实时业务。12 月 20 日，由阿里巴巴承办的 Flink Forward China 峰会在北京国家会议中心召开，来自阿里、华为、腾讯、美团点评、滴滴、字节跳动等公司的技术专家与参会者分享了各公司基于 Flink 的应用和实践经验。在大会的主题演讲上，阿里巴巴集团副总裁周靖人宣布，阿里巴巴内部 Flink 版本 Blink 将于 2019 年 1 月正式开源！阿里希望通过 Blink 开源进一步加深与 Flink 社区的联动，并推动国内更多中小型企业使用 Flink。

会上，AI 前线对阿里巴巴计算平台事业部研究员蒋晓伟（花名量仔）进行了独家专访，他与我们分享了关于下一代实时流计算引擎的看法，并针对 Blink 的重要新特性、开源后 Blink 与 Flink 之间的关系、Blink 后续规划等问题进行了解答。

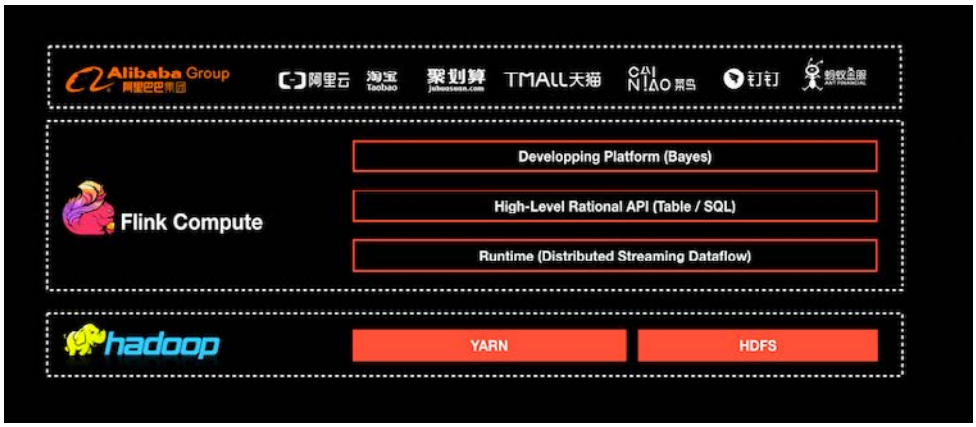
阿里巴巴与 Flink

随着人工智能时代的降临和数据量的爆发，在典型的大数据业务场景下，数据业务最通用的做法是：选用批处理的技术处理全量数据，采用流式计算处理实时增量数据。在许多业务场景之下，用户的业务逻辑在批处理和流处理之中往往是相同的。但是，用户用于批处理和流处理的两套计算引擎是不同的。

因此，用户通常需要写两套代码。毫无疑问，这带来了一些额外的负担和成本。阿里巴巴的商品数据处理就经常需要面对增量和全量两套不同的业务流程问题，所以阿里巴巴就在想：能不能有一套统一的大数据引擎技术，用户只需要根据自己的业务逻辑开发一套代码。这样在各种不同的场景下，不管是全量数据还是增量数据，亦或者实时处理，一套方案即可全部支持，这就是阿里巴巴选择 Flink 的背景和初衷。

彼时的 Flink 不管是规模还是稳定性尚未经历实践，成熟度有待商榷。阿里巴巴实时计算团队决定在阿里内部建立一个 Flink 分支 Blink，并对 Flink 进行大量的修改和完善，让其适应阿里巴巴这种超大规模的业务场景。简单地说，Blink 就是阿里巴巴开发的基于开源 Flink 的阿里巴巴内部版本。

阿里巴巴基于 Flink 搭建的平台于 2016 年正式上线，并从阿里巴巴的搜索和推荐这两大场景开始实现。目前阿里巴巴所有的业务，包括阿里巴巴所有子公司都采用了基于 Flink 搭建的实时计算平台。



目前，这套基于 Flink 搭建的实时计算平台不仅服务于阿里巴巴集团内部，而且通过阿里云的云产品 API 向整个开发者生态提供基于 Flink 的云产品支持。

以下内容整理自 AI 前线对蒋晓伟的采访。

开源的时机

AI 前线：为什么选择现在将 Blink 开源？这其中有哪些考量？什么样的时机才是开源最合适的时机？

蒋晓伟：在我看来，有几个因素：第一个因素是，这几年我们一直试图把阿里对 Flink 的改进推回社区，但社区有自己的步伐，很多时候可能无法把我们的变更及时推回去。对于社区来说，需要达成共识，才能更好地保证开源项目的质量，但同时就会导致推入的速度慢一些。经过这几年积累，我们这边和社区之间的差距已经变得比较大了。Blink 有一些很好的新功能，比如批处理功能，在社区版本是没有的。在过去这段时间里，我们不断听到有人问，Blink 什么时候能开源、是不是能开源这样的呼声。我们有两种方法，一种就是慢慢地推回去再给用户用。但我们认为这样等下去对社区不是最好的。我们还是希望尽快把我们的代码拿出来，尽量让大家都能用起来。所以最近这半年，我们一直都在准备把代码整理好去进行开源。

选择在这个时间点开源有几个好处：第一个好处是我们所开源的这些代码在阿里内部经过像双一十、双十二这样巨大流量的检验，让我们对它的质量有更大的信心，这是非常大的好处；第二个好处，Flink Forward 大会是第一次在中国举办，在这样一个场合开源表明了阿里对 Flink 社区坚定的支持，这是一个比较好的场合。主要是基于这些考虑。

选 Blink 还是 Flink？这不会是一个问题

AI 前线：开源的 Blink 版本会和阿里巴巴内部使用的 Blink 保持一致吗？

蒋晓伟：即将开源的是阿里巴巴双十二的上线版本，还会有一些小的改进。

AI 前线：Blink 开源后，两个开源项目之间的关系会是怎样的？未来 Flink 和 Blink 也会由不同的团队各自维护吗？

蒋晓伟：开源的意思是，我们愿意把 Blink 的代码贡献出来，但这两个项目是一个项目。有一件事情需要澄清一下，我们将公开 Blink 的所有代码，让大家都可以看到，但与此同时，我们会跟社区一起努力，通过讨

论决定 Blink 以什么样的方式进入 Flink 是最合适的。因为 Flink 是一个社区的项目，我们需要经过社区的同意才能以分支的形式进入 Flink，或者作为变更 Merge 到项目中。我想强调一下，我们作为社区的一员需要跟社区讨论才能决定这件事情。

Blink 永远不会成为另外一个项目，如果后续进入 Apache 一定是成为 Flink 的一部分，我们没有任何兴趣另立旗帜，我们永远是 Flink 的一部分，也会坚定地支持 Flink。我们非常愿意把 Blink 的代码贡献给所有人，所以明年 1 月份我们会先将 Blink 的代码公开，但这期间我们也会和社区讨论，以什么样的形式进入 Flink 是最合适的、怎么贡献是社区最希望的方式。

我们希望，在 Blink 开源之后，和社区一起努力，把 Blink 好的地方逐步推回 Flink，成为 Flink 的一部分，希望最终 Flink 和 Blink 变成一个东西，阿里巴巴和整个社区一起来维护。而不是把它分成两个东西，给用户选择的困难，这不是我们想要的。

因此未来用户也不会面临已经部署了 Flink、是否要把 Flink 迁移到 Blink 的问题，企业选型时也不需要在 Flink 和 Blink 之间抉择，Blink 和 Flink 会是同一个项目。Blink 开源只有一个目的，就是希望 Flink 做得更好。

Blink 改进了什么？

AI 前线：能不能重点介绍一下即将开源的 Blink 版本有哪些比较重要的新技术特性？与 Flink 最新发布版本相比，阿里的 Blink 做了哪些方面的优化和改进？

蒋晓伟：阿里巴巴实时计算团队不仅对 Flink 在性能和稳定性上做出了很多改进和优化，同时在核心架构和功能上也进行了大量创新和改进。过去两年多，有很多更新已经推回给社区了，包括 Flink 新的分布式架构等。

目前我们的 Blink 版本跟社区版本还有几点差异，第一个是稳定性方面，我们做了一些优化，在某些场景会比社区版本更加稳定，特别是在大规模场景。另外还有一个比较大的不一样是我们全新的 Flink SQL 技术栈，它在功能上，特别是在批处理的功能上比社区版本强大很多。它支持现在标准 SQL 几乎所有的语法和语义。另外，在性能上，无论是在流式 SQL 还是批 SQL，我们的版本在性能上都有很大的优势。特别是在批 SQL 的性能方面，当前 Blink 版本是社区版本性能的 10 倍以上，跟 Spark 相比，在 TPCDS 这样的场景 Blink 的性能也能达到 3 倍以上。如果用户对批处理或者对 SQL 有着比较强的需求，我们这个版本会用户可以得到很多好处。

Blink 在阿里内部的应用

AI 前线：请介绍一下 Blink 在阿里内部的使用情况。目前 Blink 在阿里的大数据架构中扮演什么样的角色？在阿里内部主要用于哪些业务和应用场景？

蒋晓伟：现在阿里的大数据平台上，所有的实时计算都已经在使用 Blink；同时，除了实时计算以外，在一些流批一体化的场景也会用 Blink 来做批处理；我们在机器学习场景也有一个探索，叫做 Alink，这个项目是对 Flink Machine Learning Library 的改进，其中实现了大量的算法，都是基于 Flink 做实时机器学习的算法，Alink 在很多场景已经被证明在规模上有很大的优势。同时，我们在图计算场景也有一些探索。

AI 前线：目前阿里内部有多少部门在使用 Blink ？

蒋晓伟：前段时间我们刚刚做过统计，阿里的技术部门大约有 70% 都在使用 Blink。Blink 一直是在用户的反馈之中成长起来的，对于内部用户反馈的数据倾斜、资源使用率、易用性方面的问题，Blink 都做了针对性的改进。

现在 Blink 用的最多的场景主要还是实时计算方面，阿里还有一些业务现在相对比较新，还没有进入实时计算的领域，等这些业务进入实时计算领域时也会使用 Blink。

在批处理方面，阿里内部也有一个自研的批处理引擎叫做 Max Compute，Max Compute 也会拥抱 Flink 生态，在语法和语义上做和 Flink 兼容的工作。未来，整个阿里的计算体系和平台都会融入同一个生态。

后续规划

AI 前线：接下来阿里对于 Blink 还有哪些规划？包括技术改进、落地应用、更新维护、社区等几个方面。

蒋晓伟：从技术上说，今天我们公布了 Flink 在批处理上的成果，接下来，我们会对技术持续投入，我们希望每几个月就能看到技术上有一个比较大的亮点。下一波亮点应该是机器学习场景。要把机器学习支持好，有一系列的工作要做，包括引擎的功能、性能和易用性。这些工作我们已经在内部的讨论和进行之中，接下来几个月，大家应该会看到一些成果。我们也在和社区讨论一些事情。除了机器学习之外，我们在图计算方面也有一些探索，包括对增量迭代更好的支持。做完这些之后，可以认为 Flink 作为大数据的计算引擎已经比较完备了。

同时，我们也重点去做 Flink 的生态，包括 Flink 与其他系统之间的

关系、易用性等。Flink 要真正做好，不仅需要它本身功能强大，还需要把整个生态做得非常强大。这部分我们甚至会跟一些 ISV 合作，看看是不是能够在 Flink 之上提供更好的解决方案，进一步降低用户的使用门槛。

在社区方面，我们希望能够把把 Blink 完全融入 Flink 社区，一起做 Flink 社区的运营，让 Flink 真正在中国、乃至全世界大规模地使用起来。

在应用方面，实时流计算其实有很多很有潜力的应用场景，但有一些可能大家不是非常熟悉，我们会对这些场景做一些推广。以实时机器学习为例，它往往能够给我们带来比一般的机器学习更大的效果提升。去年，实时强化学习给我们在搜索上带来了 20% 以上的提升。除此之外，在安全领域（比如实时的 Fraud Detection）、监控报警方面，还有 IoT 领域，实时流计算都有非常广泛的应用场景。这些 Flink 现在可能已经做了，但是大家还没有意识到，Flink 能够给大家带来这样的商业上的好处。

AI 前线：Blink 开源之后，后续阿里在这基础上做的变更和更新会以什么样的方式推回社区版本？

蒋晓伟：我们理想的方式是，阿里内部的版本是社区的 Flink 版本加上一些定制化的插件，不需要对 Flink 本身做修改，而是对 Flink 做增加。比如跟阿里内部系统交互的部分跟社区是不适用的，就会保持在内部，我们希望这些修改不动 Flink 代码，而是用插件的方式加在 Flink 上面。最终的方式就是，对于所有公司都有用的修改会在 Flink 代码本身做修改，使所有使用 Flink 的公司都能从中获利，而对接阿里内部系统的部分就只在阿里内部使用。

下一代实时流计算引擎之争

AI 前线：先在很多人提到实时流计算引擎，都会拿 Spark 和 Flink 来做对比，您怎么看待下一代实时流计算引擎之争？未来实时流计算引擎最重要的发展方向是什么？

蒋晓伟：Spark 和 Flink 一开始 share 了同一个梦想，他们都希望能够用同一个技术把流处理和批处理统一起来，但他们走了完全不一样的两条路，前者是用以批处理的技术为根本，并尝试在批处理之上支持流计算；后者则认为流计算技术是最基本的，在流计算的基础之上支持批处理。正因为这种架构上的不同，今后二者在能做的事情上会有一些细微的区别。比如在低延迟场景，Spark 基于微批处理的方式需要同步会有额外开销，因此无法在延迟上做到极致。在大数据处理的低延迟场景，Flink 已经有非常大的优势。经过我们的探索，Flink 在批处理上也有了比较大的突破，

这些突破都会反馈回社区。当然，对于用户来说，多一个选择永远是好的，不同的技术可能带来不同的优势，用户可以根据自己业务场景的需求进行选择。

未来，在大数据方向，机器学习正在逐渐从批处理、离线学习向实时处理、在线学习发展，而图计算领域同样的事情也在发生，比如实时反欺诈通常用图计算来做，而这些欺诈事件都是实时地、持续不断地发生，图计算也在变得实时化。

但是 Flink 除了大数据领域以外，在应用和微服务的场景也有其独特的优势。应用和微服务场景对延迟的要求非常苛刻，会达到百毫秒甚至十毫秒级别，这样的延迟只有 Flink 的架构才能做到。我认为应用和微服务其实是非常大的领域，甚至可能比大数据更大，这是非常激动人心的机会。上面这些都是我们希望能够拓宽的应用领域。

AI 前线：在技术方面，Spark 和 Flink 其实是各有千秋，但在生态和背后支持的公司上面，Flink 是偏弱的，那么后续在生态和企业支持这块，阿里会如何帮助 Flink ？

蒋晓伟：这次阿里举办 Flink Forward China 就是想推广 Flink 生态的重要举动之一。除了 Flink Forward China 大会，我们还会不定期举办各种线下 Meetup，投入大量精力打造中文社区，包括将 Flink 的英文文档翻译成中文、打造 Flink 中文论坛等。在垂直领域，我们会去寻找一些合作伙伴，将 Flink 包装在一些解决方案中提供给用户使用。

AI 前线：关于开源项目的中立性问题。阿里现在在大力地推动 Flink 开源项目的应用和社区的发展，但业界其他公司（尤其是与阿里在其他业务上可能有竞争的公司）在考虑是否采用 Flink 的时候可能还是会对社区的中立性存在一些疑虑，对于这一点，阿里是怎么考虑的？

蒋晓伟：阿里本身会投入非常大的力量推动 Flink 社区的发展和壮大，但我们也非常希望有更多企业、更多人加入社区，和阿里一起推动社区发展，这次阿里承办 Flink Forward China 峰会就是想借此机会让更多公司参与进来。光阿里一家是无法把 Flink 生态做起来的。我希望大家能够看到我们在做的事情，然后消除这样的疑虑。我们会用自己的行动表明，我们是真的希望把 Flink 的社区做大，在这件事情上，我们并不会有私心。

被高估的 2018：深度学习发展并没有想象的快

作者 Carlos E. Perez
译者 ambodhi

离 2019 年还剩不到 1 个月的时间了。这几年来，每个年初我都会预测当年深度学习的趋势，到年底就对预测进行回顾，今年亦不例外。本文是我对 2018 年深度学习预测的回顾。这个回顾的目的是量化深度学习的快速发展。通过回顾我自己的预测，对比新的一年的进展，我可以了解这个领域发展到底有多快。

本文将回顾我对 2018 年深度学习的预测，并对这一年的进展进行评论：

> AI前线注
> 作者对2018年深度学习的预测可参阅《10 Alarming Predictions for Deep Learning in 2018》（《关于2018年的深度学习十大惊人预测》）

1. 多数深度学习硬件初创公司都将失败

2018 年，深度学习硬件公司中，公开承认失败的公司没几家（有一家公司已宣称失败：KnuEdge），但是它们都还没有交付产品。有意思的是，尽管这些初创公司未能交付产品，却仍然能够筹集到更多的资金！

> AI前线注
> 这里有一份人工智能硬件初创公司列表

去年我的观点是，深度学习初创公司将无法正确地估计自己可以为自己的潜在的客户群提供多少可用软件的成本。随着深度学习堆栈变得越来越丰富、越来越复杂，初创公司还能迎头赶上吗？这是个疑问。

最大的失败来自 Intel。在 NIPS 2017 大会上，它们不是大张旗鼓地宣扬最终将提供硅片吗？今年，它们的 Nervana 衍生产品是 MIA。它们是否能够在明年春天之前推出“Spring Crest”？让我们拭目以待。

有多家初创公司都号称自己有可用的硅片。深度学习人工智能如此热门，每个人都想“发明”自己的芯片。简单列举一下，这些初创公司包括：GraphCore、Wave Computing、Groq、 Habana、Bitmain、Cabricon、Esperanto、Novumind、Gryfalcon、Hailo 和 Horizon。这还没算上三星、ARM、Xilinx、高通和华为等传统半导体制造商。另外有一个新情况是，现在传统的云服务供应商并不是购买 AI 芯片，而是研发自己的 AI 芯片：比如 Amazon 的 Inferentia、Microsoft 的 Brainwave、Facebook、阿里巴巴的 AliNPU 和百度的昆仑。芯片领域将很快就会变得拥挤起来！

现在的情况是，深度学习硬件由极其庞大且有能力的公司来主导：Nvidia 和 Google。虽然你无法得到 Google TPU2 的物理硬件，但你可以通过云服务对 Google TPU2 进行虚拟访问。相比之下，Nivida 提供了云计算（通过第三方云服务供应商）和硬件的两种可用选项。

能够与这两大巨头进行竞争的供应商是 AMD。AMD 支持最新版本的 TensorFlow，并且拥有逐渐成熟的深度学习堆栈。与 Nvidia 和 Google 不同，它们缺少 Tensor 核心组件（即 Systolic Array（脉动阵列））。但对更为传统的训练和推理工作来说，AMD 的硬件性能可与 Nvidia 媲美。这听上去可能会觉得没什么大不了的，但是你要知道，AMD 已经遥遥领先于任何其他初创公司的竞争对手。AMD 和 Nvidia 有着相似的规模经济，因为它们的大部分硅片都是用于日常性用途（即游戏、渲染、高性能计算等）。

老实说，这个领域有太多的追随者。成熟的深度学习市场可能只支持不超过 3 个竞争对手。Nvidia 和 Google 早已巩固它们在市场的地位，如此一来，就只剩下一个空位了！为了生存下去，公司必须使解决方案的部署尽可能简单容易、畅行无阻。此外，每家公司必须让自己的产品具有自己的特色——不要每家公司都去做图像处理！为了能够有效地提供无缝体验、同时建立起自己的市场定位，这些公司必须在软件方面进行投资（顺便说一句，我也正寻找硬件公司里的相关职位）。

2. 元学习将成为新的 SGD

元学习（Meta-learning）还没有取代 SGD（Stochastic Gradient Descent，随机梯度下降），但以神经架构搜索形式的元学习在这一领域中已经取得巨大的进步。本年的一个关键的进展与超网络（Hypernetwork）搜索有关。请参阅 Medium 上的博文：

> 《Deep Learning Architecture Search and the Adjacent Possible》（《深度学习架构搜索及相邻可能》）

我们在元学习中取得的另一个进步是在算法方面，它的灵感来自于少量学习（few-shot learning）MAML。这催生了各种算法：Reptile、PROMP 和 CAML。元学习方法需要大量的训练数据，这是因为它是在各种各样的任务中学习，而不仅仅是在不同的样本中学习。它有两个用于学习的迭代循环，一个是迭代任务的外部学习循环，一个是迭代训练数据的内部循环。基于 MAML 的方法只考虑任务的初始化，因此不需要可比较相同类型的数据。

> AI前线注
> Reptile：https://blog.openai.com/reptile/
> PROMP：https://arxiv.org/pdf/1810.06784.pdf
> CAML：https://arxiv.org/pdf/1810.03642.pdf

我现在开始认为，无监督学习和元学习实际上是同一个问题。进化解决这个问题的方法是通过踏脚石（stepping stone ）技能的发展。这意味着它完全取决于正在解决的问题的类型：是用于预测，自助控制，还是生成设计的无监督学习或元学习任务？每一种问题都需要不同的基本技能，因

此可以训练一个聚合的基本模型，而不是每一个任务都从头训练。引导是一种虚构的方法，它吸引了那些相信可以通过数学来拯救世界的研究人员。

> AI前线注
> 关于作者提及的无监督学习和元学习实际是同一个问题，可参阅论文《Unsupervised Learning via Meta-learning》（《通过元学习进行无监督学习》）

总而言之，在2018年看来最有前途的两种元学习方法是进化启发的架构搜索和少量学习MAML方法。

3. 生成模型驱动一种新的建模方式

目前，生成模型仍然主要局限于娱乐应用。当然，BigGAN 搜索创建了很多噩梦般的图像，但不幸的是，使用高保真生成模型来取代标准计算方法的研究仍然在进行。

> AI前线注
> BigGAN创建的图像点击这里可见。

DeepMind 研究蛋白质折叠已有两年，他们刚刚在 12 月份公布了研究成果：

> 《AlphaFold: Using AI for scientific discovery》
> （《AlphaFold：使用人工智能进行科学发现》）

DeepMind 训练了一个生成神经网络来编造新的蛋白质片段，这些蛋白质片段随后被用来不断提高所提议的蛋白质结构的得分。这是生成网络最令人印象深刻的用途之一，超越了图像、3D 结构或声音的美学生成。

关于该领域的详细信息，请参阅 Medium 上的博文：

> 《The Delusion of Infinite Precision Numbers》
> 《无穷精度数的迷惑》

4. 自我对决是一种自动化的知识创造

AlphaGo 引入的自我对决（Self-play）方法在应用程序使用方面上，并没有深入人心。然而在研究中，OpenAI 的 Dota Five 已经证明，自我对决可以解决一个非常困难的 AI 问题（即，即时战略游戏）。阻止这种使用的主要障碍似乎是在这种场景中框架问题的困难，以及现实问题中存在的多种不确定性。

深度强化学习（Deep Reinforcement Learning）在 2018 年遭到了很多严厉的批评，但是 OpenAI 的 Ilya Stuskever 却对 Dota Five 看似无限的深度强化学习可扩展性非常着迷，他比大多数人更早如此预测通用人工智能，见 Medium 这篇博文：

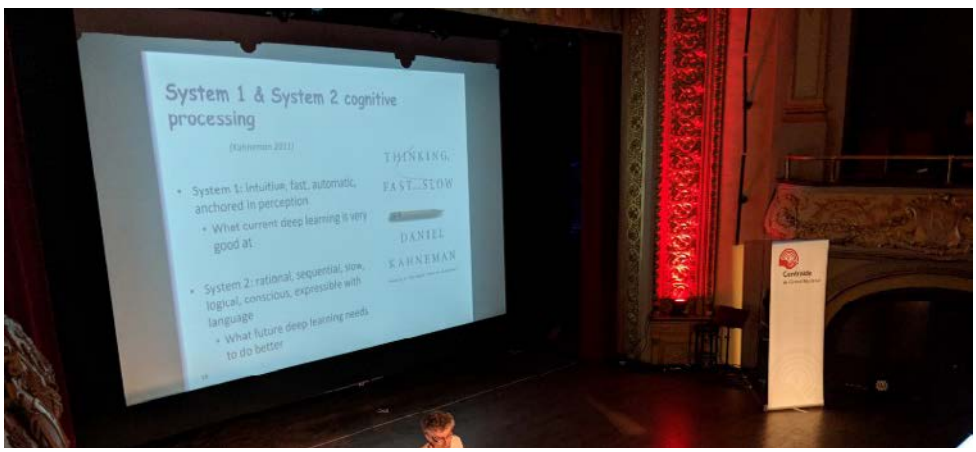
> 《Why AGI is Achievable in Five Years》
> 《为什么通用人工智能在五年内能够实现》

尽管强化学习可扩展性得到了明显的验证，仍然有一些新论文对深度强化学习的稳健性提出了质疑，如论文《Are Deep Policy Gradient Algorithms Truly Policy Gradient Algorithms?》（《深度策略梯度算法是否真的是策略梯度算法？》）

我个人倾向于支持内在激励方法而不是深度强化学习方法。原因在于，大多数难题的回报函数（reward）都是稀疏的，有些还具有欺骗性。

5. 直觉机器将缩小语义鸿沟

当 Yoshua Bengio 开始使用双重过程理论（Dual Process theory）来解释深度学习的局限性时，你就知道他在正确的轨道上。



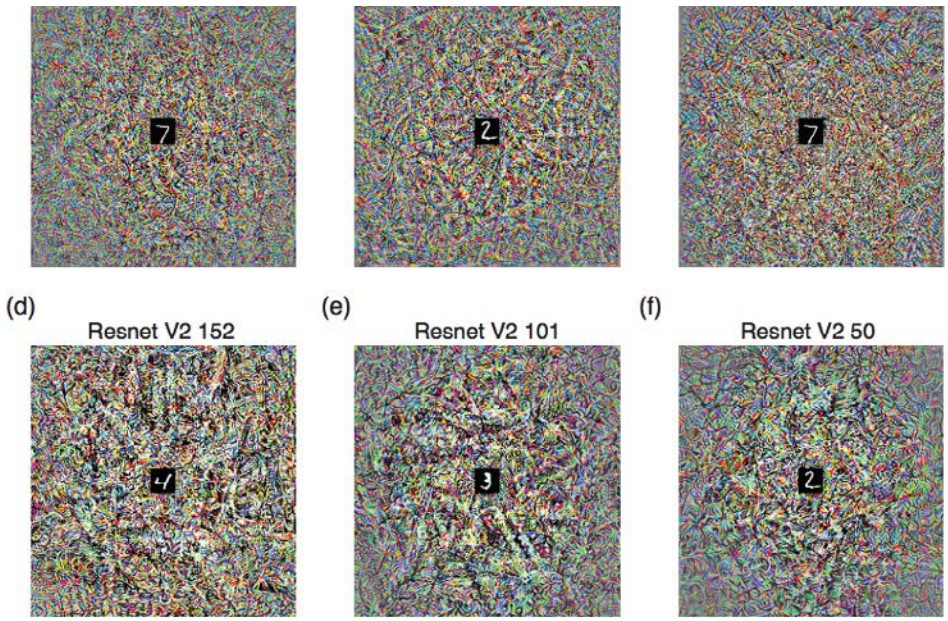
因此，深度学习机器就是人工直觉（Artificial intuition）的想法，在 2018 年已经成为主流。虽然，双重过程理论是人类思维的一个很好的模型，

但我们需要更充分地理解直觉思维的丰富性（System 1），直觉思维不仅仅是快速思考，也不仅仅是摊销推理（Amortized inference）。人类大脑中有很多东西我们还没有在深度学习中复制出来，请参阅 Medium 上的这篇博文：

> 《Where is the Artificial Ingenuity in Deep Learning?》
> 《深度学习里的人工智能在何处？》

6. 可解释性是无法实现的

现在的问题是，总体来说，我们在理解深度学习网络的本质方面并没有取得任何进展。越来越多的和我们认知相悖的研究被发布出来，打破了我们关于深度学习网络应该如何工作的理论。DeepMind 就抛出了一个打破认知的“破坏性”研究成果：



> 《Deep Learning’s Uncertainty Principle》
> 《深度学习的不确定性原理》

接着就是神经网络的对抗性重编程（Adversarial Reprogramming），论文《Adversarial Reprogramming of Neural Networks》（《神经网络的对抗性重编程》）证明了可以使用对抗特征来对神经网络进行重编程。在论

文中他们演示了如何破坏在 Imagenet 中训练的 MNIST 分类网络：

神经网络内部发生了什么，我们对其的理解仍然是远远不够的。

至于如何做出直观的解释，我认为这仍然是一个“富饶”的领域，还需要进行更多的研究。2018 年是暴露出许多深度学习局限性的一年，在这一年里，我们在开发更好的人机界面方面的进展并不大。请参阅 Medium 的这篇博文：

> 《Fake Intuitive Explanations in AI》
> （《人工智能中虚假直觉的解释》）

7. 深度学习研究信息泛滥成灾

今年，关于深度学习的论文数量翻了一番。更糟糕的是，审稿人的素质却在急剧下降。因此，杂讯比已跌至谷底。现在，每个人都需要在堆积如山的新研究论文中为自己筛选出真正有价值的研究。

有些工具，如 SemanticScholar、Arxiv Sanity 和 Brundage Bot 等论文网站都可以用来帮助你了解其中的大量信息。只是那些真正新颖的发现，却太容易成为“漏网之鱼”，让人错过。

8. 通过教学环境实现工业化

深度学习在工业化方面，并没有进展。

2018 年是人工智能结盟问题变得更加清晰的一年。

人工智能结盟对人工智能教学环境至关重要。目前有几个团队已经发布了他们的框架，这些框架可以在强化学习实验中提供更好的再现性（参见：Facebook Horizon、OpenAI Baselines、DeepMind TRFL）。

> AI前线注
> Facebook Horizon： https://code.fb.com/ml-applications/horizon/
> OpenAI Baselines： https://github.com/openai/baselines
> DeepMind TRFL： https://deepmind.com/blog/trfl/

我们已经目睹了迁移学习从虚拟环境到真实环境的进展（参见：OpenAI Learning Dexterity）。我们还见证了通过稀疏回报来教授复杂技能的进展，请参阅 Medium 上的这篇博文：

> 《How to Bootstrap Complex Skills with Unknown Rewards》
> 《如何使用未知回报引导复杂技能》

9. 会话认知

我最终描绘了一幅如何获得会话认知更为清晰的路线图。不幸的是，表中提到的“达到第三阶段”意味着必须完成这份列表中的第 5 项，而这在未来几年内不太可能发生。因此，在未来几年内人们不大可能会获得会话认知。唯一的好处就是，这种认知概念已经被认可。也就是说，它已成为一个已知的未知。

> AI前线注
> 作者提到的路线图，参阅这里

要了解其中的重要发展节点，参阅：

> 《A New Capability Maturity Model for Deep Learning》
> 《一种新的深度学习能力成熟度模型》）

10. 人工智能的道德应用

人们终于意识到这个问题了！最值得注意的是，美国加利福尼亚州议会要求，聊天机器人必须主动声明它们不是人类。

法国人感到惊慌失措，于是制定了“人类的人工智能”（AI for Humanity）计划，有关详情，请参见 Medium 这篇博文：

> 《Six Months Later, France has Formulated their Deep Learning Strategy》
> 《六个月后，法国制定了深度学习战略》

在人工智能武器化方面上，许多公司也已经划清了界限：

> 《Drawing the Ethical Line on Weaponized Deep Learning Research》
> 《为深度学习研究武器化划定道德底线》

当然，问题在于，我们的经济体系更倾向于给人工智能赋予“人格”，

而不是“制造”真正的人类。详情请参阅 Medium 这篇博文：

> 《Artificial Personhood is the Root Cause Why A.I. is Dangerous to Society》
> （《人工人格是人工智能危害社会的根本原因》）

最终，对于人工智能研究人员来说，他们必须决定自己是想一辈子向孩子们出售糖水呢，还是做一些真正有意义的事情，请参阅 Medium 上的这篇博文：

> 《Is the Purpose of Artificial Intelligence to Sell Sugar Water?》
> 《人工智能的目的是出售糖水么？》

结语

总之，回顾 2018 年，对于我年初做出的预测，现在看来，很多都做过了头。

鉴于此，我必须降低对 2019 年的期望。

我们开始意识到，在指定回报函数并基于这些回报进行优化的整个机器学习范式中，存在着重大的复杂性问题。这种学习范式只会让你的系统学会利用回报函数。很多时候，尽管看似取得了明显的进展，但底层系统学会的往往只是在测试中作弊。这是一个元级别的问题，不可能一蹴而就。在提高课程学习质量方面，我们充其量只能取得渐进式的进步。

> 《Fooled by the Ungameable Objective》
> （《不可玩目标的愚弄》）
> AI前线注
> 其中“Ungameable”是作者生造的词，用来形容一套没有任何漏洞的规则。译者进行了意译。

请继续关注我对 2019 年的深度学习预测。

去哪儿网消息队列设计与实现

作者余昭辉

去哪儿网近日在 GitHub 上开源了其内部广泛使用的消息队列 (内部代号 QMQ)，本文从去哪儿网使用消息队列所碰到的各种问题出发探讨去哪儿网消息队列的设计与实现。

背景

2012 年，随着公司业务的快速增长，公司当时的单体应用架构很难满足业务快速增长的要求，和其他很多公司一样，去哪儿网也开始了服务化改造，按照业务等要素将原来庞大的单体应用拆分成不同的服务。那么在进行服务化改造之前首先就是面临是服务化基础设施的技术选型，其中最重要的就是服务之间的通信中间件。一般来讲服务之间的通信可以分为

同步方式和异步方式。同步的方式的代表就是 RPC，我们选择了当时还在活跃开发的 Alibaba Dubbo(在之后 Dubbo 官方停止了开发，但是最近 Dubbo 项目又重新启动了)。

异步方式的代表就是消息队列 (Message Queue)，MQ 在当时也有很多开源的选择：RabbitMQ, ActiveMQ, Kafka, MetaQ(RocketMQ 的前身)。首先因为技术栈我们排除了 erlang 开发的 RabbitMQ，而 Kafka 以及 Java 版 Kafka 的 MetaQ 在当时还并不成熟和稳定。而 ActiveMQ 在去哪儿网已经有很多应用在使用了，但是使用过程中并不一帆风顺：宕机，消息丢失，消息堵塞等问题屡见不鲜，而且 ActiveMQ 发展多年，代码也非常复杂，想要完全把控也不容易，所以我们决定自己造一个轮子。

问题

如果我们要在服务化拆分中使用消息队列，那么我们需要解决哪些问题呢？首先去哪儿网提供了旅游产品在线预订服务，那么就涉及电商交易，在电商交易中我们认为数据的一致性是非常关键的要素。那么我们的 MQ 必须提供一致性保证。

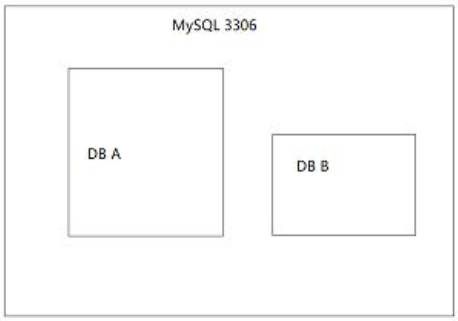
MQ 提供一致性保证又分为两个方面：发消息时我们如何确保业务操作和发消息是一致的，也就是不能出现业务操作成功消息未发出或者消息发出了但是业务并没有成功的情况。举例来说，支付服务使用消息通知出票服务，那么不能出现支付成功，但是消息没有发出，这会引起用户投诉；但是也不能出现支付未成功，但是消息发出最后出票了，这会导致公司损失。总结一下就是发消息和业务需要有事务保证。一致性的另一端是消费者，比如消费者临时出错或网络故障，我们如何确保消息最终被处理了。那么我们通过消费 ACK 和重试来达到最终一致性。

而服务端设计，在当时我们考虑的并不多，我们原计划只在交易环节中使用自己开发的 MQ，经过对未来数据的预估我们选择数据库作为 MQ Server 的消息存储，但是随着 MQ 在各系统中的大量应用，就不仅限于交易场景了，而且大家都期望所有场景中只使用一套 API，所以后来消息量迅速增长，迫使我们对存储模型进行了重新设计。再加上旅游产品预定的特征，大部分预定都是未来某个时间点的，这个时间可长可短，短的话可能是几个小时，长的话可能是半年以上，那么我们对延时消息的需求也很强烈。那么这种延时时间不固定的方式也对服务端设计提出了挑战。

接下来本文会从客户端一致性设计和服务端存储模型两方面进行讨论。

客户端一致性

提到一致性，大家肯定就想到事务，而一提到事务，肯定就想到关系型数据库，那么我们是不是可以借助关系型 DB 里久经考验的事务来实现这个一致性呢。我们以 MySQL 为例，对于 MySQL 中同一个实例里面的 db，如果共享相同的 Connection 的话是可以在同一个事务里的。以下图为例，我们有一个 MySQL 实例监听在 3306 端口上，然后该实例上有 A,B 两个 DB，那么下面的伪代码是可以跑在同一个事务里的：




```
begin transaction
insert into A.tbl1(name, age) values('admin', 18);
insert into B.tbl2(num) values(20);
end transaction
```

有了这层保证，我们就可以透明的实现业务操作和消息发送在同一个事务里了，首先我们在公司所有 MySQL 实例里初始化出一个 message db，这个可以放到自动化流程中，对应用透明。然后我们只要将发消息与业务操作放到同一个 DB 事务里即可。

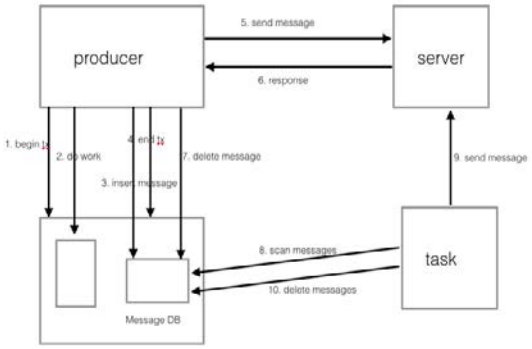
我们来看一个实际的场景：在支付场景中，支付成功后我们需要插入一条支付流水，并且发送一条支付完成的消息通知其他系统。那么这里插入支付流水和发送消息就需要是一致的，任何一步没有成功最后都会导致问题。那么就有下面的代码：

```
@Transactional
public void pay(Order order){
    PayTransaction t = buildPayTransaction(order);
    payDao.append(t);
    producer.sendMessage(buildMessage(t));
}
```

上面的代码可以用下面的伪代码解释：

```
@Transactional
public void pay(Order order){
    PayTransaction t = buildPayTransaction(order);
    payDao.append(t);
    //producer.sendMessage(buildMessage(t));
    final Message message = buildMessage(t);
    messageDao.insert(message);
    //在事务提交后执行
    triggerAfterTransactionCommit(()->{
        messageClient.send(message);
        messageDao.delete(message);
    });
}
```

实际上在 producer.sendMessage 执行的时候，消息并没有通过网络发送出去，而仅仅是往业务 DB 同一个实例上的消息库插入了一条记录，然后注册事务的回调，在这个事务真正提交后消息才从网络发送出去，这个时候如果发送到 server 成功的话消息会被立即删除掉。而如果消息发送失败则消息就留在消息库里，这个时候我们会有一个补偿任务会将这些消息从消息库里捞出然后重新发送，直到发送成功。整个流程就如下图所示：



1. begin tx 开启本地事务
2. do work 执行业务操作
3. insert message 向同实例消息库插入消息
4. end tx 事务提交
5. send message 网络向 server 发送消息
6. reponse server 回应消息
7. delete message 如果 server 回复成功则删除消息

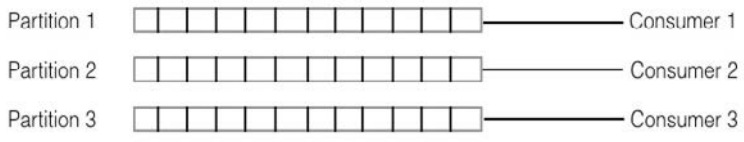
8. scan messages 补偿任务扫描未发送消息
9. send message 补偿任务补偿消息
10. delete messages 补偿任务删除补偿成功的消息

服务端存储模型

分析了客户端为了一致性所作的设计后，我们再来看看服务端的存储设计。我会从两个方面来介绍：类似 Kafka 之类的基于 partition 存储模型有什么问题，以及我们是如何解决的。

基于 partition 存储模型的问题

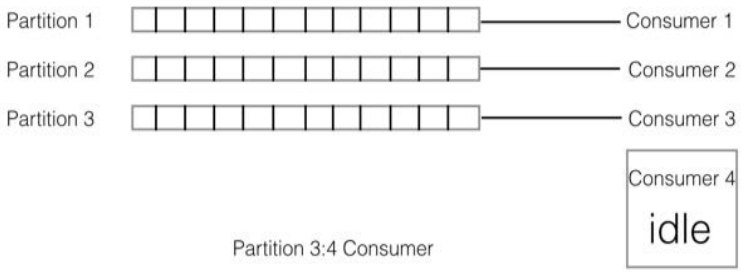
我们都知道 Kafka 和 RocketMQ 都是基于 partition 的存储模型，也就是每个 subject 分为一个或多个 partition，而 Server 收到消息后将其分发到某个 partition 上，而 Consumer 消费的时候是与 partition 对应的。比如，我们某个 subject a 分配了 3 个 partition(p1, p2, p3)，有 3 个消费者 (c1, c2, c3）消费该消息，则会建立 c1 - p1, c2 - p2, c3 - p3 这样的消费关系。



Partition 1:1 Consumer

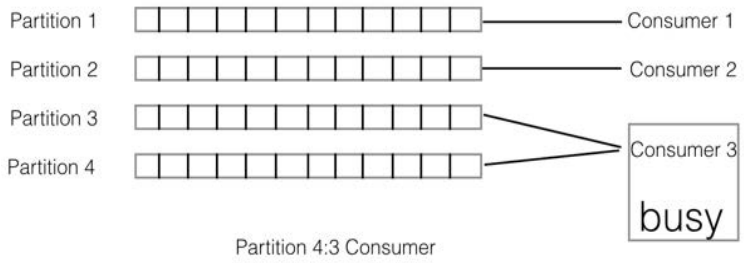
那么如果我们的 consumer 个数比 partition 个数多呢？则有的 consumer 会是空闲的。

而如果 partition 个数比 consumer 个数多呢？则可能存在有的 consumer 消费的 partition 个数会比其他的 consumer 多的情况。



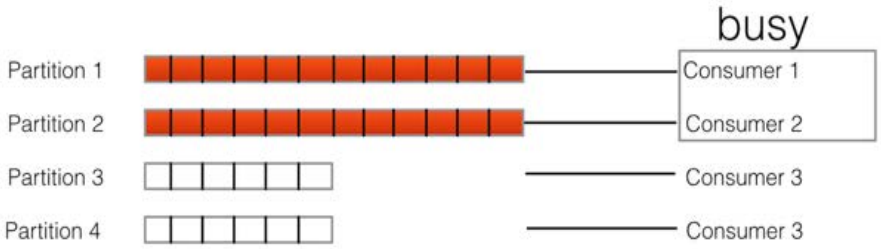
Partition 3:4 Consumer

那么合理的分配策略只有是 partition 个数与 consumer 个数成倍数关系。



以上都是基于 partition 的 MQ 所带来的负载均衡问题。因为这种静态的绑定的关系，还会导致 Consumer 扩容缩容麻烦。也就是使用 Kafka 或者 RocketMQ 这种基于 partition 的消息队列时，如果遇到处理速度跟不上时，光简单的增加 Consumer 并不能马上提高处理能力，需要对应的增加 partition 个数，而特别在 Kafka 里 partition 是一个比较重的资源，增加太多 parition 还需要考虑整个集群的处理能力；当高峰期过了之后，如果想缩容 Consumer 也比较麻烦，因为 partition 只能增加，不能减少。

跟扩容相关的另外一个问题是，已经堆积的消息是不能快速消费的。比如开始的时候我们分配了 2 个 partition，由 2 个 Consumer 来消费，但是突然发送方大量发送消息 (这个在日常运维中经常遇到)，导致消息快速的堆积，这个时候我们如何能快速扩容消费这些消息呢？其实增加 partition 和 Consumer 都是没有用的，增加的 Consumer 爱莫能助 , 因为堆积的那 3 个 partition 只能由 2 个 Consumer 来消费，这个时候你只能纵向扩展，而不能横向扩展，而我们都知道纵向扩展很多时候是不现实的，或者执行比较重的再均衡操作。



去哪儿消息队列存储模型

上面已经介绍了基于 partition 的存储模型存在的问题，那么这些问题对于我们是问题吗？或者我们的场景是否能克服这些问题呢？

现在去哪儿网的系统架构基本上都是消息驱动的，也就是绝大多数业务流程都是靠消息来驱动，那么这样的架构有什么特征呢：

- 消息主题特别多现在生产上已有 4W+ 消息主题。这是业务中使用

的消息与数据流处理中使用的最大的不同，数据流中一般消息主题少，但是每个消息主题的吞吐量都极大。而业务中的消息是主题极多，但是有很多主题他的量是极小的。

- 消息消费的扇出大也就是一个消息主题有几十个甚至上百个不同的应用订阅是非常常见的。以去哪儿酒店订单状态变更的消息为例，目前有将近 70 多个不同的系统来订阅这个消息。

结合前面对基于 partition 的存储模型的讨论，我们觉得这种存储模型不太容易符合我们的需求。

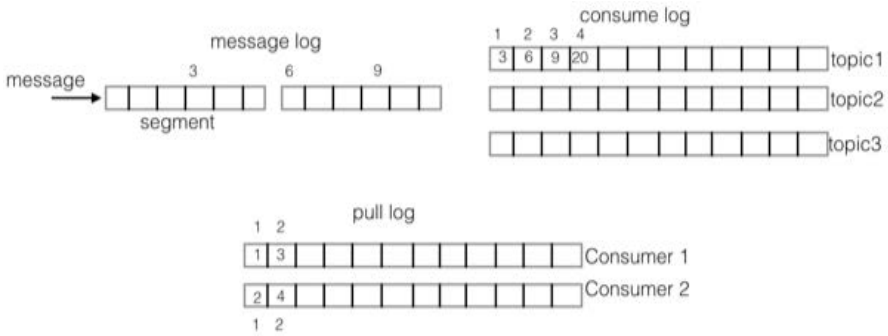
虽然我们并不想采用基于 partition 的存储模型，但是 Kafka 和 RocketMQ 里很多设计我们还是可以学习的：

- 顺序 append 文件，提供很好的性能
- 顺序消费文件，使用 offset 表示消费进度，不用给每个消息记录消费状态，成本极低
- 将所有 subject 的消息合并在一起，减少 parition 数量，单一集群可以支撑更多的 subject(RocketMQ)

在设计 QMQ 的存储模型时，觉得这几点是非常重要的。那如何在不使用基于 partition 的情况下，又能得到这些特性呢？正所谓有前辈大师说：计算机中所有问题都可以通过添加一个中间层来解决，一个中间层解决不了那就添加两个。

我们通过添加一层拉取的 log(pull log) 来动态映射 consumer 与 partition 的逻辑关系，这样不仅解决了 consumer 的动态扩容缩容问题，还可以继续使用一个 offset 表示消费进度。而 pull log 与 consumer 一一对应。

下图是 QMQ 的存储模型。



先解释一下上图中的数字的意义。上图中方框上方的数字，表示该方框在自己 log 中的偏移，而方框内的数字是该项的内容。比如 message log 方框上方的数字 :3,6,9 表示这几条消息在 message log 中的偏移。而

consume log 中方框内的数字 3,6,9,20 正对应着 messzge log 的偏移，表示这几个位置上的消息都是 topic1 的消息，consume log 方框上方的 1,2,3,4 表示这几个方框在 consume log 中的逻辑偏移。下面的 pull log 方框内的内容对应着 consume log 的逻辑偏移，而 pull log 方框外的数字表示 pull log 的逻辑偏移。

这样存储中有三种重要的 log:

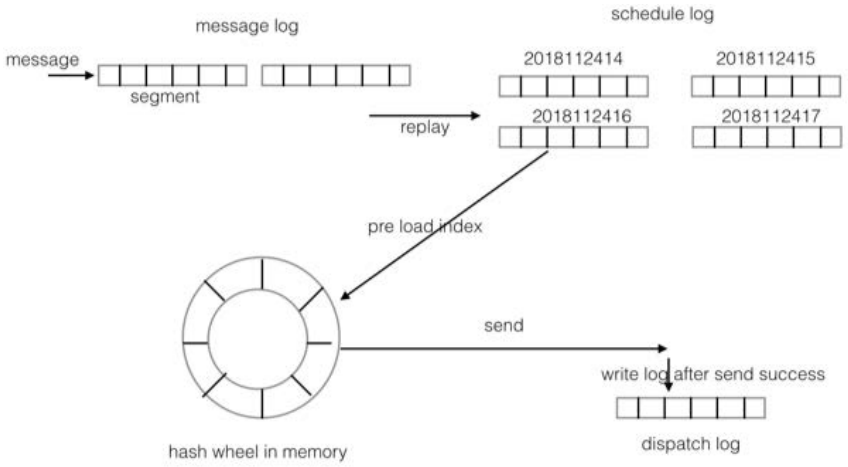
- message log 所有 subject 的消息进入该 log，消息的主存储
- consume log consume log 存储的是 message log 的索引信息
- pull log 每个 consumer 拉取消息的时候会产生 pull log，pull log 记录的是拉取的消息在 consume log 中的 sequence

那么消费者就可以使用 pull log 上的 sequence 来表示消费进度，这样一来我们就解耦了 consumer 与 partition 之间的耦合关系，两者可以任意的扩展。

延迟消息队列存储模型

除了对实时消息的支持，QMQ 还支持了任意时间的延时消息，在开源版本的 RocektMQ 里提供了多种固定延迟 level 的延时消息支持，也就是你可以发送几个固定的延时时间的延时消息，比如延时 10s, 30s…，但是基于我们现有的业务特征，我们觉得这种不同延时 level 的延时消息并不能满足我们的需要，我们更多的是需要任意时间延时。在 OTA 场景中，客人经常是预订未来某个时刻的酒店或者机票，这个时间是不固定的，我们无法使用几个固定的延时 level 来实现这个场景。

我们的延时 / 定时消息使用的是两层 hash wheel timer 来实现的。第一层位于磁盘上，每个小时为一个刻度，每个刻度会生成一个日志文件，根据业务特征，我们觉得支持两年内任意时间延时就够了，那么最多会生成 2 * 366 * 24 = 17568 个文件。第二层在内存中，当消息的投递时间即将到来的时候，会将这个小时的消息索引 (偏移量，投递时间等) 从磁盘文件加载到内存中的 hash wheel timer 上。



在延时 / 定时消息里也存在三种 log:

- message log 和实时消息里的 message log 类似，收到消息后 append 到该 log，append 成功后就立即返回
- schedule log 按照投递时间组织，每个小时一个。该 log 是回放 message log 后根据延时时间放置对应的 log 上，这是上面描述的两层 hash wheel timer 的第一层，位于磁盘上
- dispatch log 延时 / 定时消息投递后写入，主要用于在应用重启后能够确定哪些消息已经投递

总结

消息队列是构建微服务架构很关键的基础设施，本文根据我们自己的实际使用场景，并且对目前市面上一些活跃的开源产品进行对比，提出去哪儿网的消息队列的设计与实现。本文只是从宏观架构上做出一些探讨，具体的实现细节也有很多有趣的地方，目前去哪儿网的消息队列 QMQ 也已经在 GitHub 上开源，欢迎大家试用，欢迎 PR，谢谢。

2018 年终盘点：我们处在一个什么样的技术浪潮当中？

作者徐川

2018 年接近尾声，InfoQ 策划了“解读 2018”年终技术盘点系列文章，希望能够给读者清晰地梳理出重要技术领域在这一年来的发展和变化。

最近经济寒冬的说法越来越多，身边的互联网企业裁员的也有不少，越是寒冬，我们越需要了解趋势，找准前进的方向。过去几年，互联网各种“风口”此起彼伏，到底哪些才是真正的趋势？这篇文章里我将试图分析目前互联网技术的发展，找出它们背后的原因和逻辑。

如果你长期跟进本领域的前沿技术，你会发现近十年来互联网技术发生了非常大的变化，这种变化几乎在每一个领域里发生：

- 在软件架构领域，经历了从单体应用到 SOA 再到微服务；
- 在云计算领域，经历了从虚拟机到容器；
- 在数据库领域，从关系数据库到 NoSQL 再到 NewSQL；
- 在大数据领域，从批处理到流处理；

- 在运维领域，从手工运维到 DevOps、AIOps；
- 在前端领域，从 jQuery 到 React 等三大框架；
- ……

除此之外，还有一些新兴的领域如 AI、区块链，从不受重视到成为显学，开启了一波又一波的风口。

单个去看这些领域的发展，会觉得纷繁杂乱没有头绪，但如果从整体上去看，会发现它们相互之间有联系，它们的发展源于一种共同的推动力，遵循着相似的逻辑。

如果要对这个推动力、对今天这个技术浪潮起一个名字，在当前阶段我觉得可以用“云原生”，但这个短语被过度使用在各种营销语境中，它的定义会发生偏离，所以后文我不会用这个短语，而是用真正的云计算这句话。

我们当前技术浪潮的真实含义，就是我们正在走向真正的云计算时代，其它领域的发展皆由此而来，如果要更具体一点，就是：

- 云计算的技术逐渐发展成为它本来该有的模样；
- 以及与这样的云所匹配的软件架构；
- 以及与这样的架构所匹配的开发流程与方法论。

下面，我会分析几个主要的技术领域，从它们的发展历程来论述。

云计算：从虚拟化到容器到 Serverless

先从云计算说起。

2005 年亚马逊发布了 AWS，算是拉开了云计算的序幕。但是，在很长一段时间里云计算都没有兑现自己的“自动扩容、按使用付费”的宣传语。

云计算最重要的技术是分布式计算和分布式存储，分布式计算方面，最开始的技术是虚拟化，也就是所谓的“Software defined xxx”，通过对计算 / 存储和网络资源的虚拟化，同时能够给用户任意分配资源。但这里面一开始做的最好的只有文件存储这一块，AWS S3 及类似的对象存储产品给人们带来了云时代的一些实际的体验，但云服务器则还是走回了卖服务器的老路。

当然，这里的云服务器和传统服务器相比还是有优势的，至少运维不需要千里迢迢跑到机房去排查问题。但和我们想要的云服务相比还差的很远，它只是传统技术在过渡到云时代的替代品。虚拟化技术新建服务器耗时长，在扩容方面限制很大，容器技术诞生后，才终于解决了这一问题。但现在一些 MicroVM 开始出现，比如 AWS 刚刚发布的 FireCracker，试

图融合虚拟机和容器的优点，这也是当前云计算技术的一个重要关注点。

分布式存储方面，分为文件和数据库，文件通过对象存储的方式很早就解决了，数据库则面临漫长的发展过程，传统的数据库需要向分布式架构转变，同时你会发现云计算厂商成为了数据库的研发主力，这些新数据库天生就是分布式，或者天生就支持云计算特性的。

在云计算的发展过程中，有一个分支是 PaaS，最早是 2007 年推出的 Heroku，从形态上来说，它是一个 App Engine，提供应用的运行环境。PaaS 的理念被认为更贴近真正的云计算，如果你使用虚拟化的云服务器，你仍然要自己负责应用分发、部署和运维，要与各种底层接口、资源打交道，在 PaaS 上，这些都不用管了，你只需要把应用上传到云端就行。

但是，之前的 PaaS 体验较差，容易造成平台绑定，难以支持大型应用，所以并没有成为主流。这些问题直到 Kubernetes 出现后才得以解决。

在 2015 年之前，OpenStack 是云计算的主流技术，很多公司包括 IBM/ 红帽都在它身上投入重注。然而，随着曾经过分天真乐观的一些公司如思科，它们试图基于 OpenStack 进入公有云市场，但在现实面前迅速败退，以及主要参与者 Nebula 的关闭，市场的信心遭遇重挫。再加上 Docker 和 Kubernetes 的快速崛起，OpenStack 的声势已经大不如前了。

> 然而在这么多厂商的支持下OpenStack是否就无敌了呢？看似紧密的社区与厂商之间的关系，在容器这个新的技术热点面前被轻松击破。厂商不再是Pure Play OpenStack，社区贡献排名也不再提及。
>
> ——唐亚光《OpenStack七年盘点，热潮褪去后的明天在哪？》

但是，Kubernetes 还是太底层了，真正的云计算并不应该是向用户提供的 Kubernetes 集群。

2014 年 AWS 推出 Lambda 服务，Serverless 开始成为热词，从理论上说，Serverless 可以做到 NoOps、自动扩容和按使用付费，也被视为云计算的未来。但是，Serverless 本身有一些问题，比如难以解决的冷启动性能问题，因此，围绕 Serverless 的研发，以及将 Serverless 和容器技术融合也是当前的前沿课题。

> Serverless是我们过去25年来在SaaS中走的最后一步，因为我们已经渐渐将越来越多的职责交给了服务提供商。
>
> ——Joe Emison《为什么Serverless比其他软件开发方法更具优势》

架构：微服务、Service Mesh 和 Serverless

云计算为应用打造了分布式的基础设施，但是，如果应用还是以传统的单体应用的思路开发，则云计算意义并不大。

这些年里，软件架构逐渐从 SOA 进化到微服务，很多人认为微服务是一种细粒度的 SOA，在去掉了 SOA 中的 ESB 之后，微服务变得更加灵活、性能更强。但是，实施微服务需要一些前提。

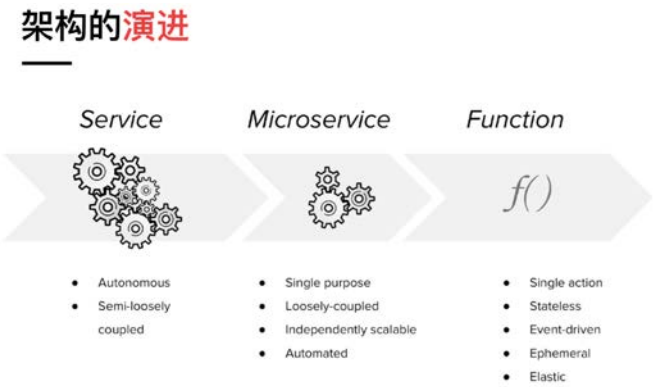
Martin Fowler 曾经总结过微服务实施的前提包括：

- 计算资源的快速分配
- 基本的监控
- 快速部署

这基本就是 Kubernetes 所起到的主要作用，虽然如 Spring Cloud、Dubbo 微服务框架在各方面已经非常完善，但随着云原生计算基金会的壮大，基于 Kubernetes 的微服务在社区中的热度越来越高，也开始有很多公司开始利用这一套技术栈来构建微服务。

到 2016 年，Service Mesh 开始引起社区的注意，Kubernetes 加上 Service Mesh，再加上 CNCF 的一些开源项目，基于 k8s 的微服务技术栈基本就完善了。2018 年 Istio 1.0 发布，更是为这股浪潮加了一把火，未来的微服务将是 k8s 和 Service Mesh 的天下。延伸阅读：Service Mesh：下一代微服务？

微服务正在逐渐走向巅峰的过程中，但它的挑战者已经出现。Serverless 或者说 FaaS 最开始只是 AWS 推出的一个功能，但随着社区和业界的跟进，逐渐有人将其认为是微服务的进化。其逻辑也很简单，从 SOA 到微服务是一个服务粒度逐渐拆分得更小的过程，FaaS 里的 Function 可以视为更小的、原子化的服务，它天然的契合微服务里面的一些理念。



许晓斌《从微服务到 FaaS》

当然，关于 Serverless 如何融入到现有架构，目前并没有成熟的经验，Serverless 本身也存在一些问题，但毫无疑问这是业界关注的重点。

数据库：从 NoSQL 到 NewSQL

在过去几年，数据库的发展同样令人瞩目。

2009 年 MongoDB 开源，掀开了 NoSQL 的序幕，一时之间 NoSQL 的概念受人追捧，MongoDB 也因为其易用性迅速在社区普及。NoSQL 抛弃了传统关系数据库中的事务和数据一致性，从而在性能上取得了极大提升，并且天然支持分布式集群。

然而，不支持事务始终是NoSQL的痛点，让它无法在关键系统中使用。2012 年，Google 发布了 Spanner 论文，从此既支持分布式又支持事务的数据库逐渐诞生，以 TiDB、蟑螂数据库等为代表的 NewSQL 身兼传统关系数据库和 NoSQL 的优点，开始崭露头角。

> 从目前已有的SQL数据库实现方案来看，NewSQL应该是最贴近于云数据库理念的实现。NewSQL本身具有SQL、ACID和Scale的能力，天然就具备了云数据库的一些特点。但是，从NewSQL到云数据库，依然有很多需要挑战的难题，比如多租户、性能等。
>
> ——崔秋《云时代数据库的核心特点》

本来事情发展到这里就结束了，但 2014 年亚马逊又推出一个重磅炸弹：基于新型 NVME SSD 虚拟存储层的 Aurora，它实现了完全兼容 MySQL（甚至连 bug 都兼容）的超大单机数据库，同时在性能上高出 5 倍以上。

延伸阅读：Amazon Aurora 读后感

另外，各种不同用途的数据库也纷纷诞生并取得了较大的发展，比如用于 LBS 的地理信息数据库，用于监控和物联网的时序数据库，用于知识图谱的图数据库等。

可以说，数据库目前处于一个百花齐放的阶段，而由于云厂商的努力，基本上新的数据库都支持自动扩容、按使用付费的云计算特征。

大数据：从批处理到流处理

Google 在 03-06 年发布了关于 GFS、BigTable、MapReduce 的三篇论文，开启了大数据时代。在发展的早期，就诞生了以 HDFS/HBase/MapReduce 为主的 Hadoop 技术栈，并一直延续到今天。在这当中，不少组件都是可

替换的，甚至有的发生了换代。这其中，最重要的换代就是处理引擎。

最开始大数据的处理大多是离线处理，MapReduce 理念虽然好，但性能捉急，新出现的 Spark 抓住了这个机会，依靠其强大而高性能的批处理技术，顺利取代了 MapReduce，成为主流的大数据处理引擎。

随着时代的发展，实时处理的需求越来越多，虽然 Spark 推出了 Spark Streaming 以微批处理来模拟准实时的情况，但在延时上还是不尽如人意。2011 年，Twitter 的 Storm 吹响了真正流处理的号角，而 Flink 则将之发扬光大。

到现在，Flink 的目光也不再将自己仅仅视为流计算引擎，而是更为通用的处理引擎，开始正面挑战 Spark 的地位。

> Apache Flink已经被业界公认是最好的流计算引擎。然而Flink的计算能力不仅仅局限于做流处理。Apache Flink的定位是一套兼具流、批、机器学习等多种计算功能的大数据引擎。在最近的一段时间，Flink在批处理以及机器学习等诸多大数据场景都有长足的突破。
>
> ——王绍翾（大沙）《不仅仅是流计算：Apache Flink®实践》序

Hadoop 本身也遭遇了 Kubernetes 的挑战。Hadoop 本身包括专用于处理大数据的编排系统如 Yarn 等，但如 Spark/Presto/Kafka 等最重要的 Hadoop 技术已经可以在 Kubernetes 上运行，使用 Kubernetes 来运行大数据技术栈，可以更好的与其它业务集成。遭遇挑战的表现之一就是 Hadoop 技术栈的两家主要提供商，Cloudera 和 Hortonworks 最近决定合并，缓慢的增长表明市场上已经容不下两家提供商了。

延伸阅读：Will Kubernetes Sink the Hadoop Ship?

运维：从手工运维到 DevOps

运维在过去几年遭遇了云计算技术的强烈冲击。那些依赖云计算提供商的公司，它们的运维的职责大大削弱，而自研云技术的公司里的运维则要求大大提高，过去的经验已经难以适用了。

这其中最重要的变化就是 DevOps 的出现，运维的身份职责发生了转变，它不再是专门跑任务脚本或者与机器打交道的人，而是变成了 OpenStack 或者 Kubernetes 的专家，通过搭建 / 管理相关的分布式集群，为研发提供可靠的应用运行环境。

DevOps 更重要的方面还是改变了应用交付的流程，从传统的搭火车模式走向持续交付，应用的架构和形态改变了，其方法论也随之而改变。DevOps 和持续交付也被认为是云原生应用的要素。

至于 AIOps 是 DevOps 在实践 AI 过程中的一些应用，称不上是范式的改变，AI 在运维领域还远远取代不了人的作用。

前端：前后端分离

前端在过去几年的变化同样称得上是翻天覆地，2008 年 Nodejs 的出现彻底激发了前端的生态，将 JavaScript 的疆域拓展到服务端和桌面，最终催生出大前端的概念。

如果纯粹看传统的前端开发的变化，不仅主流技术从 jQuery 转移到三大框架，更重要的是 SPA 和前后端分离的出现。

SPA 代表着前端的应用化，也就意味着胖客户端，部分业务逻辑可以从服务端转移到客户端完成。前后端分离更是将前端从后端独立出来，划定了领域边界。后端对前端来说，成为了数据层，只要接口能够正确返回数据，前端并不关心后端是如何做到的。

事实上，胖客户端的转变正好与后端的进化方向吻合。无论是微服务还是 Serverless，都强调无状态，这意味着你不应该用后端去生成有状态的 UI，而是让客户端自行处理状态。

延伸阅读：从前后端分离看阿里 Web 应用架构演变

延伸阅读：互联网分层架构，为啥要前后端分离？

为了应对越来越大型的客户端代码，前端发展出的技术包括 TypeScript、Redux/MobX、WebAssembly、WebWorker 等，这些也是前端重点关注的技术。

AI：互联网的新基础设施

现代的 AI 是基于大数据和机器学习的，在很多公司里大数据和 AI 属于同一个数据科学的团队。在过去两年，AI 已经用各方面的成绩证明它可以成为整个互联网的基础设施之一，帮助让我们的互联网更加的智能化。

如果把 2016 年的 AlphaGo 当做现代 AI 的起点，那么 AI 发展的历史其实很短。学术界还在研究怎么提升 AI 的算法，各个公司则是急于将 AI 应用到生产环境。

AI 从感知层大致分为两大块，一块是计算机视觉，这一块已经比较成熟，无论是人脸识别、物体检测、运动检测都已经能用于实际场景中。

另一块则是 NLP，虽然微软、Google 等宣称它们的 AI 翻译准确率已经极高，但实际上仍然不太好用，而多轮会话的问题没有解决，Chatbot 还是难以与人展开正常对话。

总之，真正的通用人工智能 AGI 离我们还远，至少现在还看不到头绪。AI 虽然在炒作中显得有些过热，但其技术和应用是真实的。

延伸阅读：致开发者：2018 年 AI 技术趋势展望

延伸阅读：被高估的 2018：深度学习发展并没有想象的快

值得注意的是，在 2018 年，国内几家涉及公有云业务的公司纷纷调整架构，将之前的云计算部门升级为智能云计算部门：

- 9 月 30 日，腾讯架构调整，新成立云与智慧产品事业群；
- 11 月 26 日，阿里集团架构调整，阿里云事业群升级为阿里云智能事业群；
- 12 月 18 日，百度调整架构，将之前的智能云事业部升级为智能云事业群。

云厂商们之所以将 AI 作为它们的顶级战略并与云计算放在一起，是因为 AI 本身需要强大的、专门定制的基础设施，是云的一个非常适合的场景；同时也因为 AI 技术有一定门槛，可以作为自身云计算差异化的一个点。总之，这些云厂商通过 AI 来卖它们的云服务。

区块链：不确定性

2018 年的区块链无疑是最有争议的话题，这里抛开那些炒作与骗局，可以看到区块链技术在 2018 年有很大的发展。

具体可分为两方面：

- 一方面是公链上一些痛点解决方案的探索和突破。包括比 POW 更好的共识机制、并发交易性能、数据存储和处理、跨链交易等等。当然，问题还远远没有得到解决。由于利益牵扯太多，这一领域也没有公认的主流解决方案。
- 另一方面是联盟链的逐渐成熟，其中代表技术为超级账本，一部分早期采用者在探索联盟链的适用场景，一部分则是做起卖水的生意，推出 BlockChain as a Service。

在现在这个时刻，区块链的未来有太多的不确定性了，无法进行预测，所以这里不再多谈。

物联网与边缘计算：为何发展不起来

物联网在过去几年一直不温不火，似乎一直在炒作中，但真正有影响

力的产品和应用比较少。曾经炒过一阵的开发板最终回归为极客的玩具。物联网本身的技术，除了各种通信协议和嵌入式操作系统和开发框架之外，近两年炒的最火的就是边缘计算了，然而，边缘计算也是炒作的重灾区。

事实上，边缘计算的定义并没有清晰，甚至连边缘是什么都没有共识。有的说终端节点、智能设备是边缘，有的说 CDN 是边缘，有的说路由器、交换机是边缘，还有的说未来的 5G 基站是边缘。

边缘计算的技术目前只看到一个 EdgeX Foundry，然而在该项目里目前还看不到一个有代表性的重量级的技术，更多是一些厂商抢占风口的占位行为。

为什么会这样呢？其实好理解，因为物联网是一个很好预测的未来趋势。

从互联网到移动互联网，是一个不断扩张的过程，不但终端节点大量增加，而且每时每刻都在线，如果将这个逻辑延伸一下就是物联网了，终端从智能手机变成任何可联网的设备。

正因为这是大家都看得到的趋势，所以所有的厂商都提前在物联网布局，试图成为下一个领先者。

但互不退让的结果，就是陷入三个和尚没水吃的境地。历史上，NFC 移动支付和物联网通信协议都有这种遭遇：

- NFC 方面，在中国，银联主推 miniSD 卡的 NFC 方案，而运营商主推带 NFC 的 sim 卡，而手机厂商更愿意将 NFC 功能直接集成至手机中。在国外，美国三大运营商推出基于 NFC 的移动支付功能 Isis，苹果谷歌各自有自己的 NFC 钱包，而 Android 阵营的手机也多半将 Android Pay 功能替换为自家的支付功能。
- 物联网通信协议方面，WiFi、蓝牙、RFID、ZigBee，背后代表了不同的利益方，而在包括工业物联网等行业之后，各种私有通信协议多达数十种。

正是因为物联网协议标准的争夺，到现在我们都没有办法简单的将两个任意两个支持联网的设备相互连接，物联网无法形成像 iOS 和 Android 一样的平台。可以想象，物联网的发展还任重而道远。

智慧城市是物联网之集大成者，然而其概念从诞生到现在数十年了，我们没能看到一个成功的落地案例。

所以，物联网的发展不会像移动互联网一样一蹴而就，而是通过在共享单车上的应用，这样一个个案例积累起来逐渐进入我们的生活。

从当下的技术看未来

看了上面的盘点，你会发现云原生或者说真正的云计算是我们当下互联网技术发展的大趋势，在这个大趋势之下，推动不同的领域进行相应的发展。

其中的代表技术，就是机器学习、Kubernetes、Serverless，它们是当下这个时代技术发展的主旋律，如果你认同这个观点，你可以得出这样一个预测：

传统的应用开发将走向以容器、Serverless 为代表的真正的云计算，而随着终端和云的更深度的集成、物联网的发展、智能化的提升，云和端的界限会变得模糊，我们和理想中的互联网会更加接近。

> 信息技术的革命将把受制于键盘和显示器的计算机解放出来，使之成为我们能够与之交谈，与之一道旅行，能够抚摸甚至能够穿戴的对象。这些发展将变革我们的学习方式、工作方式、娱乐方式——一句话，我们的生活方式。
>
> ——尼葛洛庞帝《数字化生存》

《数字化生存》是 1996 年出版的，对于理想的互联网以前我们只是凭空的想象，而现在我们知道通过怎样的技术发展路径能抵达这个理想。

技术的本质与技术发展的逻辑

技术在不断的推陈出新，令人眼花缭乱，但如果抓住了这些技术的本质，会发现太阳底下并没有新鲜事。

如果将上面的各个领域的重要技术变革提炼一下，会发现其中的一些有共同点：

- 虚拟化：将硬件资源虚拟为软件资源，然后进行统一调度和管理。
- 隔离：从虚拟机到容器，再到虚拟机与容器融合，隔离的技术定义了云的形态。
- 解耦：无论是后端的微服务、前端的前后端分离、组件化等等，都是将关注点分离，解耦合的过程。
- 编排：大量不同的服务、任务，让他们组成一个整体，相互间能良好的协作。
- 智能化：让服务个性化，或者让自动化替代以前需要人工完成的事情。

- 实时化：计算和处理在极短时间内完成，从而实时的给予反馈。

当然，其中会有一些遗漏，或者有些你并不认同，但我想表达的是，这些技术存在一些共同的本质，它们是不同的领域技术发展的共同逻辑。

再进一步：是什么在推动软件的发展？

上面我们已经知道了软件的常规发展趋势，可是，如何预测软件的颠覆式创新？要预测这个，我们需要更加深入去挖掘软件进步的源头。

软件并不是凭空发展起来的，它必须要运行在各种硬件上，软件的发展，也离不开硬件的支持。

或者说，正是硬件的不断升级和变革，支撑了软件的发展进步。云计算的诞生，正是源于大型机已经无法支撑高并发，才让人们转而采用一般硬件和虚拟化、分布式的软件技术。

软件的颠覆式创新，一定是在硬件支持的基础上，随着现有的软件架构对现有硬件能力的挖掘，再发生颠覆的可能性已经较小了。

当然，这并不是说不存在，如 Docker 和比特币的诞生，都没有利用特别新的硬件能力，更多的是现有软件发展积累到一定程度的质变。

但软件创新更多的可能性，则在于硬件的颠覆上。

AWS 推出的 Aurora 数据库就是一个很好的例子，它的诞生正是基于非易失性存储技术的重大进步。现在的趋势是，硬件的创新体现在软件上的时间会越来越短。

英特尔、英伟达研发的最新芯片，也都会被云厂商第一时间订购，充分利用硬件升级带来的性能提升。

最近，还有一个新趋势是软件厂商反过来驱动硬件的进步，谷歌、阿里、华为等都开始自研用于云和终端的芯片。

如果要预测软件的发展，我们不能不去看硬件可能带来的提升，这里我们从软件运行需要的三大资源入手：

- 计算：AI 对于计算的特殊需求，催生了相关芯片的研发。而更多非通用性芯片将推动物联网和边缘计算的发展。而在远处忽隐忽现的量子计算，一旦能普及，也必将产生颠覆。
- 存储：Nano Flash 类非易失性存储还有提升的空间，在云和端的利用也没有普及。如果非易失性存储能在内存领域有所突破，对于软件架构必将带来另一次颠覆。
- 网络：网络方面，WiFi 技术即将进入第六代，带来拥挤场合的大幅性能提升；蓝牙进入第五代，连接距离将提升至 300 米；更重

要的则是 5G，相较于 4G 数百倍的数据传输速度和低至几毫秒的延时，让很多应用都有了更大的想象空间。

对于技术发展的总结基本就到这里了。

选择技术是有风险的，如果是一家做 To B 或者 To C 的公司，选择了非主流的技术，只是会演变成长期的技术负债，但如果是一家面向开发者的云计算公司，选择错了技术则几乎注定了之后的衰落，无论是坚持下去还是切换成主流技术，都会因为错过最佳时机而步步艰难。这也是近年来新技术受到追捧的一个原因。

这种现象也导致了技术迭代的速度越来越快，开发者只要几年不关注新技术，就有一种被世界抛弃的错觉，于是每个人都很焦虑。

我希望用这篇文章，帮助你梳理技术的发展，知道正在发生什么，以及将会发生什么。只要知道了这些，想必不会那么焦虑了。

当然，由于个人能力所限，文章中不免有错漏之处，欢迎讨论交流。

2019 年 Vue 学习路线图

作者 Anthony Gore 译者无明

如果你是 Vue 开发新手，可能已经听过很多行话术语，比如单页面应用程序、异步组件、服务器端渲染，等等。你可能还听说过与 Vue 有关的一些工具和库，比如 Vuex、Webpack、Vue CLI 和 Nuxt。

浸没在术语和工具的浩瀚海洋中难免会令人感到沮丧，但其实并不是只有你一个人有这种感受，所有经验水平的开发人员都会持续感觉到这种莫名的压力。

一口气吃不成胖子，试图一下子学习所有东西可能是徒劳的，所以我将在这篇文章中展示一个高级“知识地图”，它包含了与 Vue 开发相关的关键领域，你可以使用这张地图作为 2019 年学习 Vue 的图鉴。

0. JavaScript 和基本的 Web 开发

如果我要你学习中文书籍中所写的内容，你首先要学会中文，对吧？

同样，Vue 是一个用于构建 Web 用户界面的 JavaScript 框架。在开始使用 Vue 之前，你必须了解 JavaScript 和 Web 开发的基础知识。

1. Vue 基本概念

如果你是 Vue 开发新手，应该专注于 Vue.js 生态系统的核心，包括 Vue 核心库、Vue Router 和 Vuex。

这些工具将被用在大多数 Vue 应用程序中，并为本文中提到的其他领域提供了一个构建框架。

Vue 核心功能

从根本上说，Vue 用于同步网页和 JavaScript。实现这一目标的关键特性是反应式（reactive）数据，以及指令和插值等模板功能。这些东西在一开始就要学习。

要构建你的第一个 Vue 应用程序，你还需要知道如何在网页中安装 Vue，并了解 Vue 实例的生命周期。

组件

Vue 组件是独立的可重用 UI 元素。你需要了解如何声明组件，以及如何通过 prop 和 event 在它们之间发生交互。

了解如何组合组件也很重要，因为这对使用 Vue 构建健壮、可伸缩的应用程序来说至关重要。

单页面应用程序

单页面应用程序（SPA）架构通过单个网页实现传统多页面网站一样的功能，而且不会在每次用户触发导航时重新加载和重建页面。

在将“页面”构建为 Vue 组件之后，就可以使用 Vue Router 将每个“页面”映射到一个唯一的路径，Vue Router 是一个用于构建 SPA 的工具，由 Vue 团队维护。

状态管理

随着应用程序变得越来越大，SPA 页面中会有很多组件，管理全局状态变得很困难，而且随着 prop 和 event 监听器的增加，组件变得越来越臃肿。

一种称为“Flux”的特殊模式可以将数据保存在可预测且稳定的中央存储中。由 Vue 团队维护的 Vuex 库可以帮助你在 Vue.js 应用程序中实现 Flux。

2. 现实世界中的 Vue

以上的知识可用于构建高性能的 Vue 应用程序，但如何将它们部署到生产环境中？

如果你想将基于 Vue.js 的产品发送给真实用户，你还需要了解更多东西！

项目脚手架

如果你经常构建 Vue 应用程序，你会发现几乎每个项目都需要提供配置、设置和开发者工具。

Vue 团队维护了一个叫作 Vue CLI 的工具，让你可以在几分钟内启动一个强大的 Vue 开发环境。

全栈或认证的应用程序

真实的 Vue 应用程序通常是由数据驱动的用户界面。数据通常来自使用 Node、Laravel、Rails、Django 或其他服务器框架开发的 API。

这些数据可能是由传统的 REST API 或 GraphQL 提供的数据，也可能是通过 Web 套接字提供的实时数据。

你还需要了解将 Vue 集成到完整技术栈中常用设计模式，以及确保 Vue 应用程序用户数据的安全性。

测试

如果你想开发出可维护且稳定的 Vue 应用程序，需要对它们进行测试。

在 Vue 应用程序中，可以通过单元测试来确保你的组件能够为给定输入（即 prop 或用户输入）提供相同的输出（即重新渲染的 HTML 或发出的事件）。

Vue 团队维护了一个叫作 Vue Test Utils 的工具，用于测试单独的 Vue 组件。

优化

当你将应用程序部署到远程服务器并且用户通过慢连接访问它时，它与你在开发环境中测试的速度和效率是不一样的。

为了优化 Vue 应用程序，我们可以采用各种技术，包括服务器端渲染，也就是在服务器端执行 Vue 应用程序，然后输出 HTML 页面并传给用户。

其他优化手段还包括使用异步组件和渲染函数。

3. 关键的相关工具

到目前为止，我们所看到的一切都来自 Vue.js 核心，或来自生态系统中的工具。但 Vue 不是孤立存在的，它只是前端技术栈中的一层。

高级 Vue 开发人员不仅需要熟悉 Vue，还需要熟悉每个 Vue 项目的关键工具。

现代 JavaScript 和 Babel

Vue 应用程序可以使用 ES5 开发，ES5 是几乎所有浏览器都支持的 JavaScript 标准。

要获得增强的 Vue 开发体验，并利用新的浏览器功能，你可以使用最新的 JavaScript 标准 ES2015 和 ES2016 或更高版本提供的功能来构建 Vue 应用程序。

不过，如果你选择使用现代 JavaScript，就需要提供一种支持旧版浏览器的方法，否则你的产品可能无法为大多数用户提供服务。

要实现这一目的，需要使用 Babel。它的作用是在应用程序发布之前将你的现代功能“转换”（翻译和编译）为标准功能。

WebPack

Webpack 是模块捆绑器，如果你的代码跨越了不同模块（例如不同的 JavaScript 文件），Webpack 可以将这些零散的代码“构建”到浏览器可读的单个文件中。

Webpack 还可以作为构建管道，你可以在构建代码之前对代码进行转换，例如使用 Babel、Sass 或 TypeScript，还可以使用一系列插件来优化你的应用程序。

很多开发人员觉得 Webpack 难以掌握，配置起来也很麻烦，但如果没有它，将无法使用 Vue 的一些有用的功能（如单文件组件）。

最近发布的 Vue CLI 3 提供了一种用于在 Vue 项目中抽象和自动配置 Webpack 的解决方案。

这是否意味着你不需要学习 Webpack 了？当然不是，因为你仍然不可避免地需要进行定制或调试 Webpack 配置。

TypeScript

TypeScript 是 JavaScript 语言的超集，为我们提供了类型（String、Boolean、Number 等），这样我们就可以编写健壮的代码，并尽早发现错误。

Vue.js 3 将于 2019 年推出，将完全使用 TypeScript 编写。但这并不意味着你一定要在你的 Vue 项目中使用它，但如果你想要为 Vue 贡献代码，或者想要理解它的内部工作原理，就需要了解 TypeScript。

4. Vue 的框架

构建在 Vue 之上的框架让你无需从头开始实现服务器端渲染，还可以创建自己的组件库以及完成很多其他常见任务。

有很多很好的 Vue 框架，在这里我们只列出使用最为广泛和最重要的三个框架。

Nuxt.js

如果你想要构建一个高性能的 Vue 应用程序，就需要基于组件的路由、服务器端渲染、代码拆分和其他尖端的功能。你还需要像 SEO 标签这样的功能。

Nuxt.js 通过各种社区插件提供了这些开箱即用的功能，以及更多的功能选项，如 PWA。

Vuetify

谷歌的 Material Design 是一个使用十分广泛的指南，用于构建漂亮的逻辑用户界面，并被用在谷歌的产品（如 Android 和 Web）当中。

Vuetify 在一系列 Vue 组件中实现了 Material Design。因此，你可以使用 Material Design 布局和样式快速构建 Vue 应用程序，以及模态、警报、导航栏、分页等小部件。

NativeScript-Vue

Vue.js 是一个用于构建 Web 用户界面的库。如果你想将它用于原生移动界面，可以使用 NativeScript-Vue 框架。

NativeScript 是一个用于在 iOS 和 Android 上使用原生用户界面组件构建应用程序的系统，而 NativeScript-Vue 是一个基于 NativeScript 的框架，提供了 Vue 的语法和组件的使用方式。

5. 杂项

在最后一部分，我们将介绍其他一些内容。

插件开发

如果要在项目中重用 Vue 功能或为 Vue 生态系统做贡献，可以将功能作为 Vue 插件来开发。

动画

如果你需要使用动画，请了解一下 Vue 的过渡系统，它也是 Vue 核心的一部分。你可以在向 DOM 添加元素或从 DOM 中删除元素时应用动画。

你需要创建 CSS 类来定义所需的动画效果，无论是淡入淡出、更改颜色还是你喜欢的其他方式。当向 DOM 中添加元素或从 DOM 中删除元素时，Vue 会检测到这些变更，并在过渡期间添加或删除相应的 CSS 类。

渐进式 Web 应用程序

渐进式 Web 应用程序（PWA）就像普通的 Web 应用程序一样，只是加入了改进的用户体验。例如，PWA 可能包括脱机缓存、服务器端渲染、推送通知等。

大多数 PWA 功能可以通过 Vue CLI 3 插件或使用 Nuxt.js 等框架添加到 Vue 应用程序中，但你仍然需要了解一些关键技术，包括 Web App Manifest 和 ServiceWorker。

全球 6 大数据中心，日均 10 亿日志场景下的高可用实践

作者 张磊、姜冰

写在前面

近几年互联网服务安全事故的频发，使得越来越多的企业及开发者意识到数据备份、灾备等重要性，高可用性、高容灾性及高可扩展性的系统和架构建设工作也被更多地置于重心。

在这个过程中，基于公有云提供的基础设施实现高可用已成为多数技术团队的选择。

而面对跨区域 + 高可用需求时，许多开发者却犯了难，一方面业务场景的不同对架构设计提出了不尽相同的需求，基本上没有可完全复制的架构模版；另一方面是纳入考虑的因素在不断增多，数据延迟、成本开销、

物理隔离、快速的恢复能力……如何能在加法的基础上做好减法，对技术团队创新能力提出了更高要求。

对此，InfoQ 专访了 FreeWheel 架构师张磊及 FreeWheel 首席工程师姜冰，探讨在服务全球 6 个数据中心，日均产生近 10 亿广告投放展示日志的场景下，做好跨区域高可用的实践之道。

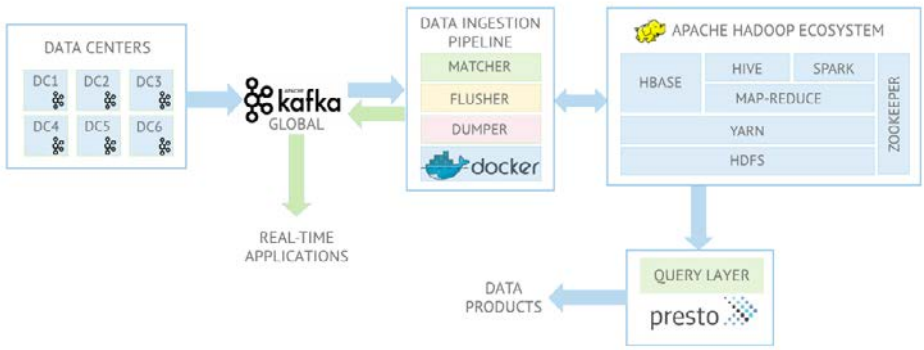
跨区域、高可用性实践背景

数据处理平台提出的多维挑战

作为美国 Comcast 旗下领先的视频广告管理和投放平台，FreeWheel 的数据来自于全球 6 大数据中心（美国东西海岸各两个、欧洲两个），每天产生 10 亿左右的广告投放展示日志，新增超过 3TB 的数据，并且有至少 18 个月的数据可以随时满足查询寻求。

在数据应用对实时性和可靠性要求逐渐增加的情况下， FreeWheel 根据其自身数据特点和应用需求，采用了一套以 Kafka，Hadoop 和 HBase 为核心的 Lambda 处理架构。

其中，各个数据中心的广告服务器实时地将广告数据传输到本地 Kafka 集群。中心化的 Kafka MirrorMaker 将多数据中心的数据聚集到一个全局的 Kafka 集群。流式处理框架会从全局 Kafka 集群消费数据，处理之后一方面写回 Kafka（为下游的各类实时应用服务）；一方面写入 HBase，并由 Hadoop 离线作业将 HBase 的内容聚合转换成 Parquet 文件写入 HDFS。构建于 Presto 之上的 OLAP 服务会同时查询 HBase 和 HDFS，对外提供数据的实时或批量查询（整体架构见下图）。



FreeWheel 数据处理系统架构图

FreeWheel 部署的这一套数据处理系统，很好地实现了预期的设计目标。但过往几年里，随着数据量的不断增长和业务变化的需求，这种模式

面临了如下挑战：

1. 可扩展性：越来越多的数据产品接入到新的数据平台，对数据处理和查询服务的扩展性提出了严苛的要求。而在自建的数据中心，受限于规划和整体容量，很难根据需求灵活地扩展。同时，在“超级碗”这样大型赛事的直播带来流量瞬时波动的场景下，传统的数据中心也很难满足对容量的弹性需求。
2. 高可用性：虽然像 Hadoop 这样的大数据基础设施本身可以保证一定程度的高可用性，但在遇到数据中心整体性故障时，依然会对服务造成影响。
3. 开发和测试效率：大数据平台开发和测试中，基于真实环境的集成测试和性能测试可以覆盖单元测试和回归测试的盲区，这需要经常启停一些基础组件甚至整套端到端服务。但在自有数据中心里，受到资源限制，很难做到及时满足需求，因而也降低了开发和测试的效率。

理论上，上述挑战中的一部分可通过在另一个自建数据中心里再部署一套数据处理系统解决或缓解，但考虑到规划、扩容所需时长以及问题根治性等因素，在 2016 年底，FreeWheel 决定与 AWS 合作，将数据平台和数据处理系统部署到云端。

选型过程并不复杂，FreeWheel 主要从成熟程度、数据监管、可用区（AZ）数以及原生服务的数量与广度等方面考量，最终选择了 AWS 作为云服务商。

基于 AWS 原生服务的使用权衡

整体上，FreeWheel 在 AWS 上也部署了一套 Kafka Mirror Maker，同步所有数据中心的数据，再将数据处理平台基本原样搬到了 AWS 上。

但如何权衡是直接使用像 Kinesis、DynamoDB 这样的 AWS 原生服务，还是自己维护 Kafka、HBase 等基础设施？ FreeWheel 也有一些思考。

张磊对此总结了两点。一是从平台需求出发。AWS 许多原生服务在某种程度上的确能带来开发和维护成本的降低，但对于基础数据平台这类数据量极其庞大并且对稳定性要求很高的业务，AWS 的原生服务能不能满足需求，能满足需求的情况下成本是不是可控，这些都会是最终影响选型的因素。二是需要考虑开发者自身对技术的把控。AWS 的原生服务很多时候对使用者来说还是黑盒。当大家需要花大量时间去了解这些服务的内部运作原理，限制和故障处理时，不一定能降低时间和运维成本。

涉及到具体权衡与改造，姜冰也举例讲解了何时该选择自身维护：

以 AWS 原生服务 Amazon EMR 和 Amazon Kinesis 为例，映射到 FreeWheel 的数据处理系统中相应的是 Hadoop Cluster 和 Kafka。基于 FreeWheel 以 7*24 小时流计算为主的业务类型，如果直接使用 EMR 及 Kinesis，将面临如下问题：

- 原生服务开放程度较弱，因而开发者缺乏更多的机会维护和管理自身集群。例如从对外兼容性上看，Kafka 下游接了多样的开源解决方案，与 Spark、Flink 等有天然集成，但 Kinesis 是闭源的托管服务，下游集成由 AWS 主导，日志记录的更新和变化将受制于 AWS 内部研发；
- EMR 适合批处理的工作模式，在全天候不间断的运行状态下，基于 AWS 基础计算和存储资源自建的 Hadoop 集群能够提供更好的可用性，从而更好地应对流处理场景。

在上述场景下，FreeWheel 并没有主动选择 EMR 或 Kinesis 原生服务，而是从成本、性能、开放程度、规模化和容错管理等维度与自身维护做了对比实践。但张磊也提到，在一般的使用场景下，对于 AWS 的原生服务，比如 RDS，Athena 等，FreeWheel 会鼓励并尽可能地使用，这样才能更有效地满足高可用需求，并降低总体开发维护成本。

截至目前，经过测试和调整，FreeWheel 已逐步把面向客户的所有产品都迁移到 AWS 环境中，目前自有数据中心里只有极少数对内的服务在运行。而最终它们也都会迁移到 AWS 环境中。

高可用实践需求及实现方案解析

高可用性设计目标

一个完整的全系统高可用性设计通常会涉及五大层面：基础设施、服务器、数据、应用程序、系统服务。

在基础设施层，FreeWheel 更多的是将 AWS 视为一项服务。虽然 AWS 会充分保证其基础设施的稳定性，FreeWheel 还是会在所有的服务都可能出问题这一假设下指导设计。服务器层也是如此。

数据层，如果该数据存在于 FreeWheel 自身系统，其通常会借助于像 Kafka、HBase 的天然多副本能力，即设置多个副本提供数据容灾。

应用程序层，FreeWheel 一般会将它做到无状态，从而更容易实现快速恢复和高可用，无论是水平扩展还是垂直扩展都更方便。

服务层，FreeWheel 采用的是 Failover（故障转移）机制。即应用服务器设置多台，彼此之间通过心跳联系。数据库、缓存、应用服务器等任

何位置出现瓶颈就只需增加处理能力。目前，对于服务器的对等可替换性，主要有三种类型：Active-Active（双主）、Active-Standby（主备）以及单 Active。

- 在双主模式下，FreeWheel 通常会通过先绑定 Amazon ELB，再注册到 Amazon Route 53 的方式，从而实现自动对应用无感知访问。
- 对于主备模式，FreeWheel 通常会将相应的信息注册到 ZooKeeper，由 ZooKeeper 实现 Failover 的分布式协调服务，从而实现 Master 的选举和对外服务发现。
- 对于单主模式，一般是一些离线任务，FreeWheel 会将其绑定在 ASG（Auto Scaling Group）中，出现问题时可以自动扩展重试。

高可用的整体实现方案

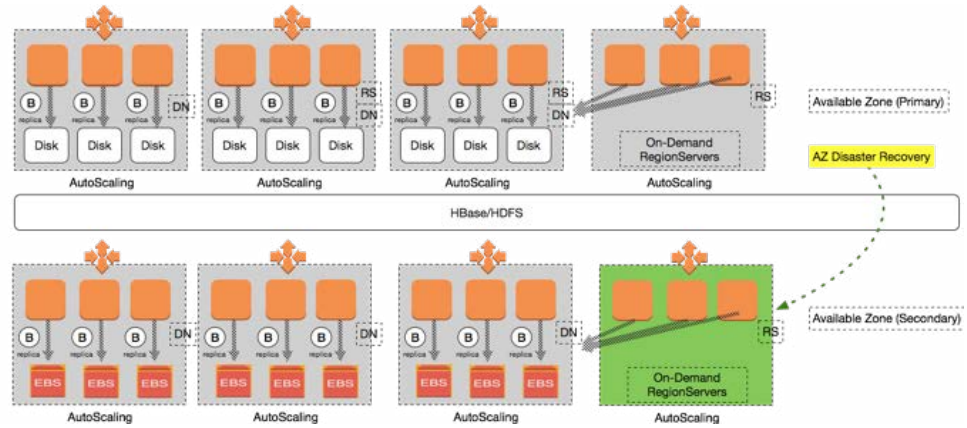
无论是跨 Region（区域）还是跨 AZ（可用区），数据交换都将产生一定的成本开销。面对着庞大数据量和跨区域 / 可用区间的数据延迟，在 AWS 多可用区实践过程中，FreeWheel 针对数据平台对于高可用的需求及 AWS 自身高可用实践经验，采用了多种定制方式——主要对 Hadoop-HBase、Kafka、Zookeeper 及数据处理部分实现了不同的高可用解决方案。

Hadoop-HBase 多可用区

在设计之初，FreeWheel 就试图平衡了成本、性能和可用性之间的关系。但是类似数据库系统中的 CAP 理论，这三者很难同时得到满足。例如，AWS 上很多及其类型和存储类型都不尽相同。为了提升 HBase 性能需要采用 instance store 这类本地 SSD 存储方式，而相应的弊端也随之产生：一旦挂载这个 instance store 这个实例出现异常，相应的存储就会丢失。尤其是当其真实宿主机发生大面积故障时，可能导致数据的彻底丢失。

因此，FreeWheel 也需要对 HBase 在多个可用区上实现高可用。

对于存储模块，FreeWheel 选择两个可用区来部署，通过 HDFS Storage Policy 接口，实现数据块副本数在主可用区和次可用区的分布。对于无状态的数据应用优先部署在与 Hadoop / HBase 主可用区同侧，并支持在次可用区快速部署。应用和数据优先同侧的策略，可以有效降低数据应用的因为跨可用区带来的数据延迟，并降低可用区之间网络传输的成本开销。借助 AWS 提供不同存储介质的 IOPS 和持久化特点，实现针对不同业务的工作负载灵活选择存储节点配置，并实现业务之间的物理隔离（原理如下图）。



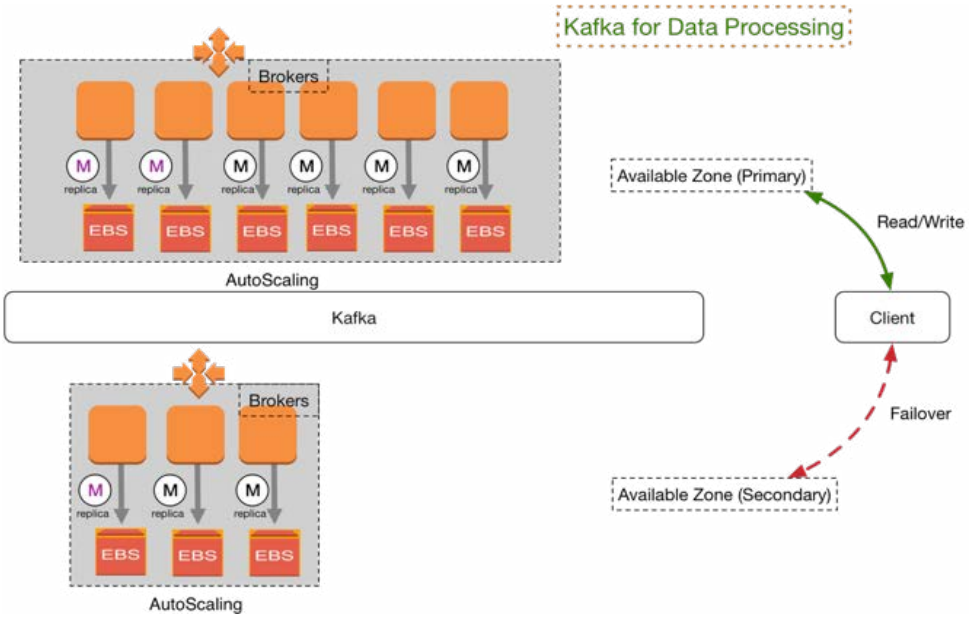
其中，隔离思路主要是基于不同的工作负载选用不用的 EC2 实例或 EBS 实例。这样也可根据自身的业务特点，在 AWS 中选择不同的硬件、实例类型或 EBS 类型来满足需求。此外，FreeWheel 也可以在 AWS Hadoop 上实现一组机器的上线及下线，而且只把具有某一类标签的机器上、下线，而不影响到其他数据。

Kafka 多可用区

Kafka Topic 内每个 Partition 有多个副本，多可用区部署，需保证在不同可用区的副本个数的划分。姜冰提到，在正常情况下每个 Partition 里可设置多个副本，如果要保证高可用，意味着需要将多个副本同时被部署到同一可用区上，但同时，这样做的弊端是无法保证可用区级别高可用，如果一个可用区宕机将导致整个 Partition 不可用。

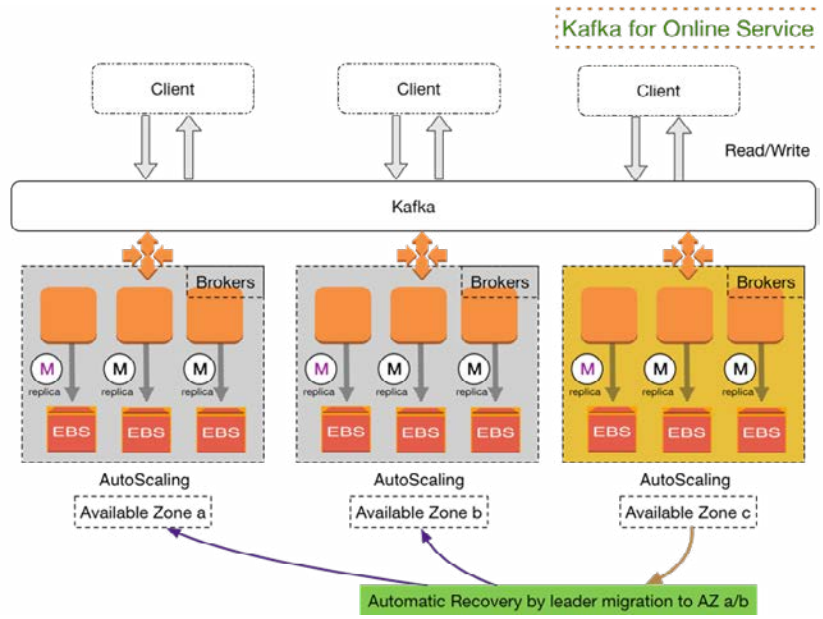
于是，考虑到跨可用区带来的数据延迟以及成本开销，FreeWheel 针对数据应用和规模实现了不同的策略。

一种策略是，对于用于数据处理流程的 Kafka Topic，在次可用区仅部署 Partition 副本中的 Follower。应用程序和 Kafka 主可用区同侧，读写数据的流量全部在同一可用区完成，这样在主从可用之间，仅有 Leader 和 Follower 之间的网络传输开销。在主可用区发生故障时，从可用区的 Follower 切换成 Leader 对外提供服务（原理如下图）。



在这种情况下，Kafka 消费者只会在同一可用区消费，从而避免跨可用区间因网络传输带来的成本消耗，同一可用区间的延迟性也大大降低。

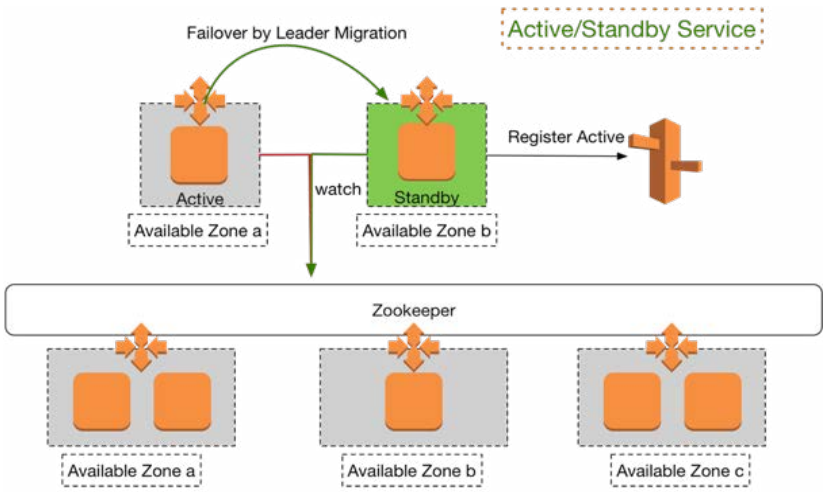
而对于以在线服务为主的数据，为了提供更强的高可用和更快速的恢复能力，Kafka 采用 3 可用区对等部署的方案，无差别的对外提供服务：每一个 Partition 的 Leader 和 Follower 以公平的策略随机分配到不同的可用区，在 AWS 出现可用区级别的故障时，Kafka 借助 Leader 和 Follower 之间的切换，保证服务的高可用（原理如下图）。



放在 FreeWheel 的具体业务场景中，为保证广告主预算与广告竞价间反馈回路的可用性高且延时性低，避免出现广告超量投放或投放价格与预算偏差等问题，FreeWheel 即需采用上述第二类这种解决方案。

Zookeeper 多可用区部署方案

Zookeeper 集群是由 1 个 leader 和若干个 follower 组成，这些节点被部署到 3 个 AWS 可用区。应用通过名字注册服务将主工作服务（Active）注册到 Zookeeper。在可用区发生故障时，应用 Active / Standby 角色根据 Zookeeper 名字服务状态变化进行切换，并注册最新的 Active 服务到 Route53（原理如下图）。



数据处理高可用部署方案

目前 FreeWheel DIP（Data Ingestion Pipeline）架构由两部分组成，一是流处理工作类型，二是上下游以小时为级别的批处理工作类型。

流处理消费一次来源于 Kafka，同时会和 HBase 交互，所以就流处理来说，如果能解决数据和状态高可用，其本身是属于一条无状态的数据处理流水线。反观，这也是为什么 FreeWheel 会花更多功夫在 Kafka、Hadoop-Hbase 和 Zookeeper 高可用部署上的原因，从而进一步确保流式数据处理状态和高可用。

流式处理主要通过 Hadoop YARN 部署，采用 ASG 做扩展 / 收缩，以及采用 ASG 绑定同一 YARN 队列的方式来保证高可用性。

对于批处理工作类型的解决方案，FreeWheel 也有比较好的 Failover 方案应对异常状况下的自我修复。总的来说，利用分布式的天然架构，FreeWheel 可以通过监控措施很快地对其进行恢复。

高可用实践之切换部署

因为通过对有状态数据（诸如 Kafka、HBase、Zookeeper ）实现多可用区，从底层向上的各个业务可形成自我恢复的能力，即使在发生故障时，也已能在极端的情况下为用户无缝提供完整服务。

当跨可用区级别切换时，FreeWheel 更多贯彻的是 DevOps 理念，即不用专职的运维人员，通过监控和脚本自动化的工具和方式保证服务高可用。目前在 FreeWheel 内部受重视的一种模式是监控和报警的代码化，即通过相应的框架实现监控和报警，用代码做管理，而非以往那种在站点上做简单配置的方式。

对于监控和报警，FreeWheel 用到的主要工具包括 Datadog 及开源工具如 TerraForm（类似于 AWS Cloud Formation）和 SaltStack 等。

Datadog 的工作方式是在每一台需要监控的服务器上运行它的 Agent。FreeWheel 主要用 Datadog 实现 EC2 级别监控，这样可以基于机器类型、用途等，分门别类对其服务进行监控，一是实现系统级别（如 CPU、内存、磁盘、网络 I/O ）的相关监控，二是实现基于应用级别的监控。

另外，TerraForm 会应对启动资源、扩展 / 回收和权限配置之类的场景，从而实现 AWS 资源配置。SaltStack 实现配置的幂等性管理，并针对机器 tag 进行分布式调用和部署检查。

当可用区级别需切换时，目标是希望能做到让用户无感知。但目前如果需从主区域切换至备用区域情况下，FreeWheel 当前的设计方案是允许有一定（分钟级别）的宕机时间。因为在一般情况下，一个区域 3 个或 3 个以上的可用区（ FreeWheel 只会使用有至少 3 个可用区的区域）已经足够保证高可用性。当整个区域出现整体性故障时，允许有一定的宕机时间是综合成本和需求的整体考虑。

高可用实践收效及未来规划

张磊在采访中提到，伴随着高可用实践，FreeWheel 也获得了业务的高弹性。对于“超级碗”及世界杯这类大型体育赛事开始时，整个架构会有更好的扩展能力应对突发流量，帮助其在 AWS 上实现高度的动态扩展性。

未来，FreeWheel 一方面会尝试将一部分系统走向容器化，比如 Data Ingestion Pipeline。当然，对于 HBase、Hadoop 如何更好地结合 K8S 或利用 Amazon EKS 等工具，FreeWheel 还需要下一阶段的调研及实验。

另一方面，针对更大范围的高可用、数据安全等问题，FreeWheel 还将持续探索怎样在平衡性能和成本的条件下多个区域提供服务和快速的灾

难修复的能力。

受访嘉宾

张磊，FreeWheel 架构师，现负责公司数据平台和数据产品的整体技术把控。

姜冰，FreeWheel 首席工程师，现全面负责大数据平台的架构和研发工作。

UCloud 大规模数据中心网络管理系统建设之路

为了应对大规模数据中心的网络配置自动下发、运维效率提升、混合云网络实时打通等需求，UCloud 团队研发上线了物理网络编排器（以下简称编排器）。它是网络运维自动化建设的基础应用系统，为网络运维人员提供一个简单易用的网络设备配置工具，使之从繁琐的交换机命令编写中获得解放，并具备足够的准确性、稳定性和安全性。

基于此系统，数据中心建设时上万规模级别 IDC 的网络建设周期由原先的 2-3 天缩短至 2-3 小时，且上线成功率也从此前的 80% 提高到 99%。上线至今，该系统已成功服务于 3000 余台交换机，维护了 100 条以上的运营商专线。其所承载的网络接入吞吐达 200G，DCI（数据中心

内部互联）吞吐接近 2T 左右，并能支持混合云业务在用户控制台的网络实时打通等（物理接线除外）。

难点分析

传统网络部署主要通过人工配置的方式实现。当网络达到一定规模时，采用人力堆砌的方式效率低下不说，大量的配置命令，也容易导致较高的上线错误率。结合公司目前的情况，在分析业界开源的配置下发方案后，我们决定采用基于 Python with NAPALM (Python module) 方案自建一套业务配置下发系统，这套系统的内部 Codename 为 XUANWU（中文玄武）。

NAPALM 模块底层的配置下发系统（比如 Ansible、Paramiko、Salt）虽然是通用的，但目前只有主流的厂家才支持对应的库，一些二线品牌的厂家支持的并不完善，需要自研。寄希望厂家提供丰富和标准的配置下发 API 是不太切实的奢望，我们要结合业务开发一套适用于 UCloud 的配置下发系统。

通过分析，我们发现，如果要开发一套基于业务的网络编排器，必须要解决以下障碍：

1. 同一个底层，命令不同的设备厂家提供的配置命令不一样。如创建一条静态路由，同样一个需求，锐捷的命令是“ip route 0.0.0.0 0.0.0.0 1.1.1.1”，而华为的命令是“ip route-static 0.0.0.0 0.0.0.0 1.1.1.1”；
2. 同一款设备型号，由于软件版本不同，实现同一个功能的命令组合不同。如华为的 CE6851-48S6Q-HI 设备，同样实现 tunnel 创建这功能，当版本 <=V1R2 时，tunnel 口需要绑定一个 service_type 为 tunnel 的聚合口来激活，而之后的版本，就不需要绑定了；
3. 不同厂家设备，实现同一个需求的配置模式不一样。如将交换机物理口划入某聚合口配置需求下，华为只需要将物理口绑定抽象出来的聚合口即可继承聚合口下所有属性，而 H3C 则要在物理口下将聚合口的属性配置再配置一遍。
4. ……

这些形形色色、大大小小的人能解决的问题却让自动化举步维艰，要想实现网络配置自动化，首先必须将这些问题总结抽象成具体的场景，然后通过分级归类来规避差异。

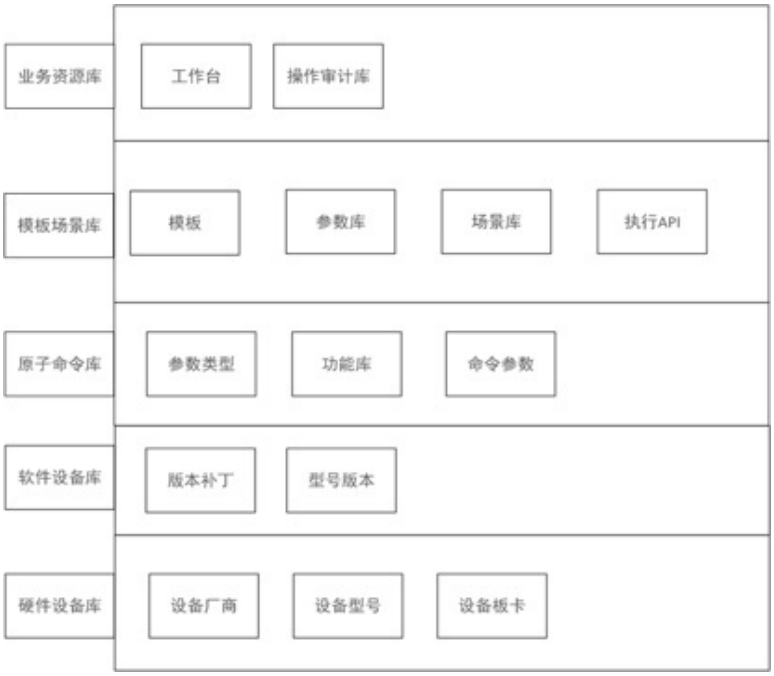
编排器架构及业务模型搭建

经过内部多次的讨论和归纳，我们发现真正业务调用网络的，是一个一个具体的场景，这些场景能够实实在在满足具体需求，如一个业务的需

求是打通托管到公有云的路由，那么实际上网络实现的就是静态路由的下发，静态路由下发就成了一个配置场景。

这里，需要注意的是场景是不关心设备厂家的，因此就会衍生出两个问题：1、配置命令的差异；2、配置模式的差异。我们需要将这两层抽象出来处理，经过测试和抽象，我们设计了业务模型的网络配置下发系统：

图 1 UCloud 基于场景的网络配置下发系统架构图



- 第一步：构建原子命令类。由厂商，设备型号，型号版本，版本补丁构成一个原子命令的类，该类原子命令为一个录入模板。
- 第二步：原子命令录入。先录入功能，再录入功能对应的原始设备命令。
- 第三步：创建模板。将一个或者多个原子命令拖拽构建成能对应业务需求的模板。
- 第四步：创建 API。为模板建立对应的 KEY 值，使其能够为灵活的以 API 的方式被业务调用。

以一个具体的 API 创建过程为例，首先录入设备的软硬件版本，将软件和硬件组合为一个单位，按照原子命令规范抽象出一层 GROUP 组，每一个 GROUP 为一个原子命令录入标准，接下来按照命令功能为单位关联一条或者多条原子命令。

具体构成关系图如下：

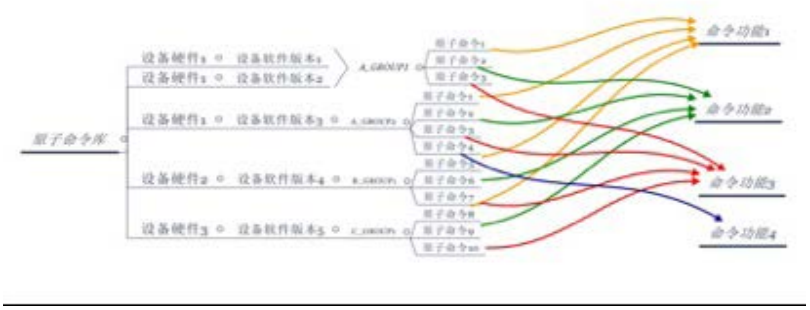


图 2 命令功能 - 原子命令 - 版本 group- 软硬件设备对应表

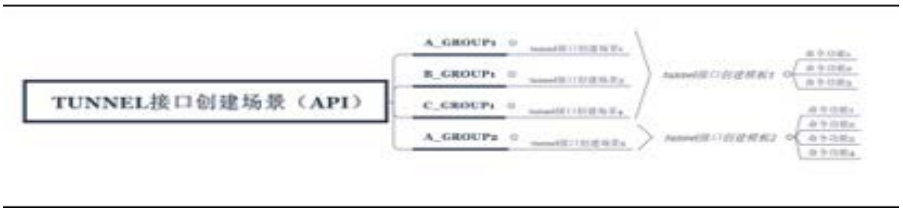


图 3 API- 场景 - 模板 - 命令功能对应表

场景中涉及到的命令功能创建完成后，通过前端编排创建模板，并结合实际属性自动生成场，由此 K-V 的 API 模型基本就落成了。但是此时的创建只能关联一个具体的设备类型的场景，如果其它场景有其它设备，API 就无法生效了。因此还需要将多个关联了 GROUP 组的场景加入大场景组中以大 API 的方式暴露出去，至此该 API 就能兼容多厂家的配置命令了。

设计与优化实践

考虑到现网的特殊性，需要编排器具备较高的准确性、稳定性和安全性。为满足以上特性，整个系统在架构设计、代码开发过程中也做了系列调优实践。

1. 基于 Kafka 的命令执行队列设计

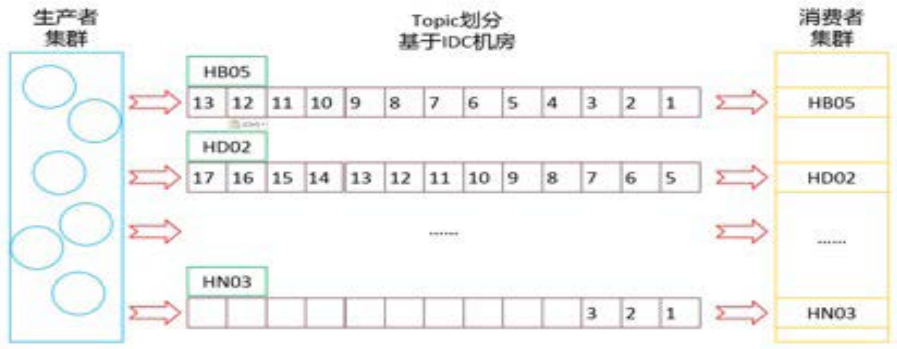
编排器需要面向公司所有在用机房按照一定的业务应用场景（以下简称场景）进行交换机配置命令的下发。以 IDC 机房作为空间维度，配置命令的执行先后顺序作为时间维度，每个场景的执行必须做好时序控制，兼顾空间和时间维度。

为了保障配置指令的准确执行到位，我们采用了基于 Kafka 的命令执行队列设计，Kafka 消息队列的主题（Topic）分类特性和消息队列生产消

费特性可以很好的兼顾指令下发队列模块对时序控制的需求。主题分类特性可用于处理 IDC 机房的空间维度，消息队列生产消费特性则对应指令执行的时间维度。

在实际应用过程中，通过对业务场景进行解析，将交换机配置指令按照 IDC 机房生产至对应的 Kafka 主题，同一个 IDC 机房的配置指令将以交换机为单位，按照执行的先后顺序生产至对应的主题。

图 4 基于 Kafka 的命令执行队列设计



通过利用 Kafka 的两大特性，结合系统的业务场景解析逻辑，我们最终实现了交换机配置指令的时空维度控制，确保指令的准确送达。系统上线后运行至今，未出现过配置指令误发、错发的情况。

2. 集群、主备与主主设计的应用

作为网络运维自动化建设的基础系统，后期将面向更多部门开放部分或全部功能，因此，编排器服务的稳定性尤为重要。系统考虑了在使用集群化等高可用技术上的各种环节，制定了以下稳定性提升设计方案：

1. Zookeeper 集群（基本操作）；
2. Kafka 集群（基本操作）；
3. Kafka 消费者集群：在每个 IDC 机房部署 Kafka 消费者集群，确保配置指令消息的稳定消费并执行；
4. MySQL 数据库 1 主 2 备：数据库读写分离，主库只接受写操作，从库只接受读操作，降低主库负载，提高数据库服务稳定性；
5. Webserver 主主：提供两台 web 后台服务器，通过 Nginx 代理实现负载均衡，结合数据库的双备设计，实现 API 请求及数据库读操作的均衡分配。

通过集群化、主备、主主设计，编排器提供的 Web 服务变得更加稳定，

配置指令消息队列也更加可靠。系统上线后运行至今，未出现过因 web 服务器或消息队列宕机导致的服务不可用情况。

3. 权限设计

系统涉及现网交换机的变更操作，每个业务场景的执行均会对现网产生影响，适当的权限管理也非常重要。系统从 2 个层面对操作者权限进行限制，并启用后端 token 认证：

- API 权限：涉及数据库的增、删、改、查，以及业务场景的下发（回滚）操作，通过 1 对 1 的 API 权限划分进行管理；
- 菜单权限：涉及菜单层级的 UI 界面操作，对前端操作页面的菜单项进行权限划分和管理；
- Token 认证：后端调用 API 时将进行 Token 认证，Token 与操作者一一对应，责任到人。

图 5 token 认证流程

通过权限认证，可以规范了使用者的操作习惯，强化使用者的安全意识，最终提高整个系统的安全性。

4. 指令下发实时反馈

传统人工配置模式下，每一位网络运维工程师必须要登录到对应的交换机才能进行配置指令下发，同时观察交换机回显信息进行指令执行结果确认。为了重现这一过程并提供更友好的信息提示效果，系统集成了指令执行结果实时展示功能，并提供执行结果详细信息查询，供网络运维工程师参考决策。

具体实现方式为，编排器前端界面在业务场景执行过程中，系统提供两种方式的执行结果信息展示：

1. 执行状态：将每一条指令的执行状态分为下发成功、下发失败、即将回滚、回滚成功、回滚失败以及登录失败 6 种，并在业务场景下发过程中通过不同边框颜色进行实时展示。

图 6 指令执行状态说明

图 7 指令执行状态显示界面

2. 指令执行回显信息：大部分指令执行之后交换机都会有回显信息提示，系统自动抓取交换机回显信息并反馈到前端，为操作者提供参考。

图 8 指令执行回显信息展示

这种图形颜色及回显信息展示，可以实时反馈配置指令的执行效果，为网络运维工程师提供参考，在提高自动化水平的同时提升操作反馈体验。

5. 原子命令库的建立

公司在用交换机主要有 HW、H3C 和 RUIJIE 三大品牌，各大品牌交换机之间配置指令存在较大差别。同时，每个品牌下面的交换机又可分为

若干型号，不同型号之间部分指令也存在一定的区别。

为了兼容公司在用的所有交换机，需要对所有交换机型号进行统一管理，同时根据不同交换机型号进行配置指令录入，做好分类管理。基于以上背景及需求，系统搭建了原子命令库，原子命令库包括以下几个子库：

1. 交换机设备库：以厂商、型号、版本、补丁为分类字段，将公司在用的所有交换机进行归类整理；
2. 交换机设备组库：在交换机设备库的基础上进行分组，将配置指令相同的设备划分到同一组；
3. 原子命令库：基于交换机设备组进行原子命令录入，同时从原子命令功能维度进行分类；
4. 原子命令功能模板库：用于录入交换机命令的功能模板；
5. 原子命令参数模板库：用于录入交换机命令的各类参数模板。

通过建立原子命令库，可以将公司所有在用交换机及其适用的配置指令进行了统一分类管理。而交换机组的设计，可以将配置指令相同的交换机归入同一组，同组设备共享原子命令，极大减少了录入原子命令的工作量。此外，通过建立原子命令功能、参数模板库，可以将原子命令录入方式标准化，为业务场景编排打好基础。

6. API 开放和二次开发

系统提供了原子粒度的交换机操作 API，不仅支持现有的网络运维操作，也可作为外部调用组件支持相关的二次系统开发。同时，通过提供统一且精细分类的 API 入口，可以实现现网交换机操作的集中管理，提高现网安全性。目前已为盘古系统（UCloud 服务器自动交付系统）、UXR 项目提供底层调用 API，支撑系统开发及相关业务开展。

最后

经过联调测试，我们已将该系统应用到多个业务场景中，且都有不错的表现。通过网络自动编排系统，我们将数据中心建设业务中，上万规模级别 IDC 的网络建设周期从原先的天缩短为小时。此外，上线成功率也从原先的 80% 提高到 99%。在混合云业务打通中，虚拟网络的业务逻辑也能通过用户的控制台操作实时打通（物理接线除外）。

扫码关注InfoQ公众号

Geekbang. 极客邦科技 | InfoQ