Kafka 2.0.0 重磅发布，新特性独家解读

今天 Apache Kafka 项目的 2.0.0 版本正式发布了！距离 1.0 版本的发布，相距还不到一年。这一年不论是社区还是 Confluent 内部对于到底 Kafka 要向哪里发展都有很多讨论：从最初的标准消息系统，到现如今成为一个完整的包括导入导出和处理的流数据平台，从 0.8.2 一直到 1.0 版本，很多新特性和新部件被不断添加。但同时更重要的，关于“一个企业级的流式数据平台，到底有哪些必须的功能”这个问题，也被不断实践和理解。

本文作者，王国璋，Apache Kafka PMC，Kafka Streams 作者。分别于复旦大学计算机系和美国康奈尔大学计算机系取得学士和博士学位，主要研究方向为数据库管理和分布式数据系统。现就职于 Confluent，任流数据处理系统架构师和技术负责人。此前曾就职于 LinkedIn 数据架构组任高级工程师，主要负责实时数据处理平台，包括 Apache Kafka 和 Apache Samza 系统的开发与维护。

增强在线可进化性

就我个人而言，这几年来学到的最重要的一课，就是要永远保证一个流式数据平台的在线可进化性（online-evolvable）。

之前我曾经读到 Amazon CTO Werner Vogels 写过的一篇博客，里面就提到这一点，并且有一个精彩的比喻：搭建一个能够在不断产品升级过程中保证永远在线的数据架构，就像是驾驶着一架简单的单螺旋桨飞机起飞，然后在飞行过程中，不断换新零件和添加新引擎，直到最后升级成一架超大的空客飞机，这一切都必须在飞行中同步完成，并且坐在里面的乘客不能有任何感觉，其难度可想而知。

而关于一个流数据平台，对于在线可进化性这一点的需求尤甚：这里我也打一个比方，尽管没有 Vogels 的那样精彩——数据流就像是一个连接城市各个地区的高速公路，高速公路所连接的地方，比如一家超市、一家影院，或者一个居民小区，就如同一个企业里面大大小小的各种应用产品或者数据仓库，超市可以暂时歇业，影院可以暂时关门翻修，居民小区甚至也可以迁出人口夷平重建，就像是产品或者数据仓库都可以短暂下线升级维护。然而高速公路却很难彻底打断重搭，因为随时都有人要上路。更多时候，它只能一边继续完成输送车辆的任务，一边增设车道或者加盖匝道。就像是一个流数据平台本身，因为不会有一个零流量的时刻，所以所有的维护和升级都需要保证同步在线完成，而且期间最好没有任何用户可感知到的性能弱化或者服务差别。而在云环境下，后者显得更为重要。对于 Kafka 而言，如果一个用户没有一个安全稳定的升级路线的话，那么她就只能停留在最初的那个版本，再不会升级。

因此我们从很早以前开始注意保证在线升级的方便性，在这一次的 2.0.0 版本中，更多相关的属性被加了进来，比如 KIP-268、KIP-279、KIP-283 等等。

KIP-268：简化 Kafka Streams 升级过程

Kafka Streams 利用 Consumer Rebalance 协议里面的元数据字符串编码诸如任务分配、全局查询、版本升级相关的信息。然而，当编码版本本身改变的时候，就需要进行离线升级。比如之前

从 0.10.0 版本像更高级的版本升级的时候，用户就需要将所有的 Streams 程序下线，换上新的 Kafka 版本号，然后在全部重启。

KIP-268 利用 version prob
可以使得旧版本的任务分配者告知其他高版本的成员暂时使用旧版本的 Rebalance 元数据编码，这样就可以让用户依然能够通过 rolling bounce 在线升级 Kafka Streams 的版本。而当所有参与的成员全部升级完毕之后，最后一次 rebalance 会自动切换回新版本的元数据编码。

KIP-279：修补多次 Kafka 分区主本迁移时的日志分歧问题

在升级 Kafka 版本或者做定期系统维护的时候，用户往往需要进行连续的多次 Kafka 分区迁移。在这次发布中我们修补了一个在此过程中可能会出现的一个会导致日志分歧发生的边缘情况。具体方案就是将此前版本中已经加入的主本 epoch 信息扩散到 OffsetForLeaderEpochResponse。如此所有主副本就可以清晰知道自己到底处于当前分区备份的哪一个阶段，从而杜绝因为消息不对等而可能导致的日志分歧。

KIP-283：降低信息格式向下转换时的内存消耗

在一个多客户端组群的环境下，客户端与服务器端的版本不匹配是常见现象。早在 0.10.0 版本中，Kafka 已经加入了允许不同版本客户端与服务器交互的功能，即高版本的 Kafka 客户端依然可以与低版本的服务器进行数据传导，反之亦然。然而当低版本的消费者客户端和高版本的服务器进行交互时，服务器有时需要将数据向下转换（format down-conversion）成为低版本客户端可以认知的格式后才能发回给消费者。向下转换有两个缺点：

- 丢失了 Kafka 数据零拷贝（zero-copy）的性能优势；
- 向下转换需要额外的大量内存，在极端情况下甚至会导致内存溢出。

前者无法避免，但是后者依然可以改进：在即将发布的 2.0 版本中，我们使用了一种新的基于分块（chunking）的向下转换算法，使得需要同时占据的内存需求大幅缩减。这使得高低版本的客户端与服务器之间的交互变得更加有效。

更多的可监控指标

对于企业级数据平台来说，另一个很重要的要求就是提供各种实时的监控能力。在 LinkedIn 的时候，同事间流传着据传是我们公司传奇人物 David Henke 的一句话：what gets measured gets fixed，充分体现了监测的重要性。

长期以来，Apache Kafka 社区不断地完善各种区块的各种指标，这每一个新添加的指标背后都有一个我们曾经踩过的坑，一段在线调试和修 bug 的痛苦经历。

举一个具体的例子：Kafka 长期以来被诟病添加分区太慢，因此在 1.1.0 版本里面来自六个不同企业的贡献者共同完成了重新设计 Kafka 控制器（Kafka Controller）这个规模巨大的 JIRA。在这个长达九个月的项目里，被谈论很多的一点就是如何增添控制器操作的各种指标。在未来更多的新功能和新属性，比如继续增强 Kafka 的伸缩性，包括

多数据中心支持等等，如何能够让用户继续便捷地实时监测这些新增功能的性能，及时发现可疑问题，并且帮助缩短需要的在线调试时间，都将是讨论的重要一环，因为这也是任何一个企业级流数据平台必须要注意到的。

在 2.0.0 版本中，我们进一步加强了 Kafka
的可监控性，包括添加了很多系统静态属性以及动态健康指标，比如
KIP-223、KIP-237、KIP-272 等等。

KIP-223：加入消费者客户端的领先指标

在此前， Kafka 消费者客户端已经加入了对每一个消费分区的延迟指标（lag metrics），定义为当前消费者在分区上的位置与分区末端（log-end-offset）的距离。当此指标变大时，代表消费者逐渐跟不上发布的速度，需要扩容。我们发现，当分区 renteion 时间很短而导致消费者跌出可消费范围时（out-of-range），此指标不能完全针对潜在的危险为用户报警。

因此在即将发布的 2.0 版本中，我们加入了另一个“领先”指标（lead metrics），定义为分区首端（log-start-offset）与消费者在分区上的位置距离，当此指标趋近于零时，代表消费者有跌出可消费范围因而丢失数据的危险。值得称赞的是，这个新特性是由国内的社区贡献者完成的。

KIP-237：加入更多 Kafka 控制器的健康指标

在完成了针对增强 Kafka 控制器性能的全面重设计之后，我们认为现在 Kafka
已经可以支持千台机器，百万分区级别的集群。在此之前，这样规模集群的一个阻碍就是 Kafka 控制器本身处理各种 admin 请求，诸如主本迁移、话题扩容、增加副本等等时的时间消耗。在完成了这个重设计之后，我们也相应地加入了更多和新设计相关的健康指标，包括控制器内部请求队列的长度，请求处理速率等等。

更全面的数据安全支持

最后，也是最近一段时间被讨论最多的，就是数据安全的重要性：在云计算大行其道的今天，多租户方案已成为潮流。而许多数据保护条例的实施，比如 GDPR，更是让“保证数据不泄漏”成为一个标准流数据平台的基本要求。这项要求包括从写入，到处理，到导出的一些系列措施，包括认证、访问控制及授权、端到端加密等等。

举个例子，如果一个 Kafka 集群可以被一个企业里面的多个部门所共享，如何控制哪些部门可以读取或写入哪些主题、哪些部门可以创建或删除哪些主题、谁可以看见别人创建的主题、以及与 Kafka 相关的其他服务和客户端，比如应该如何保护Zookeeper、谁可以直接读写、哪些 Kafka Streams 或者 Kafka Connect 应用程序可以发起哪些管理请求等，都需要提供足够的控制手段。

在 2.0.0 版本里面，我们对此提供了一系列的改进，比如更细粒度的更细粒度的前缀通配符访问控制（KIP-290、KIP-227），支持
SASL/OAUTHBEARER（KIP-255），将委托令牌扩展到管理客户端（KIP-249），等等。

KIP-290、KIP-227：细粒度前缀通配符访问控制

在 2.0 以前，很多 Kafka 自身的访问控制（access control list）机制还是粗粒度的。比如对“创建话题”这一访问方式的控制，只有“全集群”这一种范围。也就是说，对于任何一个用户来说，我们只能或者给予其在一个集群内创建任何话题的权限——比如说，这个 Kafka 集群的运维或者可靠性工程团队（Devops or SRE），或者不给予任何话题的创建权限。但是在一个多租户环境下，我们很可能会出现需要更细粒度的控制机制。比如一个 Kafka Streams 客户端，除了读取给予的源话题之外，还需要实时创建一下内部用于 data shuffling 和 change logging 的话题，但是给予其一个“全集群”的话题创建权限又太危险。

因而在 2.0 版本中，我们进一步细粒度化了很多权限，比如 KIP-290 就加入了前缀通配符（prefix wildcard）的范围，而 KIP-227 就将这种范围加入到了单个或多个话题创建的权限中。在这个机制下，用户可以被赋予单独一个话题基于其话题名的创建权限（literal topic），也可以被赋予多个话题基于其话题名前缀匹配的创建权限，比如可以创建所有名字开头为“team1-”的话题，等等。

总结

这些总结出来的经验，很多都是在将 Apache Kafka 推广到互联网以外的领域，比如银行金融，制造和零售业内的公司时发现的。我们观察到在各种不同的领域里面，公司内部工程团队的技术倚重和深度、事务逻辑的严谨性要求、对于实时性以及成本控制的权衡偏好都各有不同，因此用户对于“流数据处理”这一名词所代表的需求都或多或少有自己独特的定义，但是以上这些特征却是共同的。因此，我觉得作为一个逐步成为标准流数据平台的开源项目，Kafka 还很“年轻”，还有很多值得我们去探索，未来可期。

关于 2.0.0 版本，我们其实还有很多新的发布特性，比如新的 Kafka Streams Scala API 帮助 Scala 用户更好的利用 Scala typing system 进行编程，Kafka Connect REST extension plugin 对于认证和访问控制 的支持等等。关于更多细节，欢迎大家阅读相关公告：https://www.apache.org/dist/kafka/2.0.0/RELEASE_NOTES.html

本文转自：Kafka 2.0重磅发布,新特性独家解读