01 | 消息引擎系统ABC

胡夕 · Kafka核心技术与实战

你好，我是胡夕。欢迎你来到“Kafka 核心技术与实战”专栏。如果你对 Kafka 及其背后的消息引擎、流处理感兴趣，很高兴我们可以在此相聚，并在未来的一段日子里一同学习有关 Kafka 的方方面面。

毫无疑问，你现在对 Apache Kafka 一定充满了各种好奇，那么今天就允许我先来尝试回答下 Kafka 是什么这个问题。对了，先卖个关子，在下一期我还将继续回答这个问题，而且答案是不同的。那么，Kafka 是什么呢？用一句话概括一下：**Apache Kafka 是一款开源的消息引擎系统**。

倘若“消息引擎系统”这个词对你来说有点陌生的话，那么“消息队列”“消息中间件”的提法想必你一定是有所耳闻的。不过说实话我更愿意使用消息引擎系统这个称谓，因为消息队列给出了一个很不明确的暗示，仿佛 Kafka 是利用队列的方式构建的；而消息中间件的提法有过度夸张“中间件”之嫌，让人搞不清楚这个中间件到底是做什么的。

像 Kafka 这一类的系统国外有专属的名字叫 Messaging System，国内很多文献将其简单翻译成消息系统。我个人认为并不是很恰当，因为它片面强调了消息主体的作用，而忽视了这类系统引以为豪的消息传递属性，就像引擎一样，具备某种能量转换传输的能力，所以我觉得翻译成消息引擎反倒更加贴切。

讲到这里，说点题外话。我觉得目前国内在翻译国外专有技术词汇方面做得不够标准化，各种名字和提法可谓五花八门。我举个例子，比如大名鼎鼎的 Raft 算法和 Paxos 算法。了解它的人都知道它们的作用是在分布式系统中让多个节点就某个决定达成共识，都属于 Consensus Algorithm 一族。如果你在搜索引擎中查找 Raft 算法，国内多是称呼它们为一致性算法。实际上我倒觉得翻译成共识算法是最准确的。我们使用“一致性”这个字眼太频繁了，国外的 Consistency 被称为一致性、Consensus 也唤作一致性，甚至是 Coherence 都翻译成一致性。

还是拉回来继续聊消息引擎系统，那这类系统是做什么用的呢？我先来个官方严肃版本的答案。

根据维基百科的定义，消息引擎系统是一组规范。企业利用这组规范在不同系统之间传递语义准确的消息，实现松耦合的异步式数据传递。

果然是官方定义，有板有眼。如果觉得难于理解，那么可以试试我下面这个民间版：

系统 A 发送消息给消息引擎系统，系统 B 从消息引擎系统中读取 A 发送的消息。

最基础的消息引擎就是做这点事的！不论是上面哪个版本，它们都提到了两个重要的事实：

消息引擎传输的对象是消息；

如何传输消息属于消息引擎设计机制的一部分。

既然消息引擎是用于在不同系统之间传输消息的，那么如何设计待传输消息的格式从来都是一等一的大事。试问一条消息如何做到信息表达业务语义而无歧义，同时它还要能最大限度地提供可重用性以及通用性？稍微停顿几秒去思考一下，如果是你，你要如何设计你的消息编码格式。

一个比较容易想到的是使用已有的一些成熟解决方案，比如使用 CSV、XML 亦或是 JSON；又或者你可能熟知国外大厂开源的一些序列化框架，比如 Google 的 Protocol Buffer 或 Facebook 的 Thrift。这些都是很酷的办法。那么现在我告诉你 Kafka 的选择：它使用的是纯二进制的字节序列。当然消息还是结构化的，只是在使用之前都要将其转换成二进制的字节序列。

消息设计出来之后还不够，消息引擎系统还要设定具体的传输协议，即我用什么方法把消息传输出去。常见的有两种方法：

点对点模型：也叫消息队列模型。如果拿上面那个“民间版”的定义来说，那么系统 A 发送的消息只能被系统 B 接收，其他任何系统都不能读取 A 发送的消息。日常生活的例子比如电话客服就属于这种模型：同一个客户呼入电话只能被一位客服人员处理，第二个客服人员不能为该客户服务。

发布 / 订阅模型：与上面不同的是，它有一个主题（Topic）的概念，你可以理解成逻辑语义相近的消息容器。该模型也有发送方和接收方，只不过提法不同。发送方也称为发布者（Publisher），接收方称为订阅者（Subscriber）。和点对点模型不同的是，这个模型可能存在多个发布者向相同的主题发送消息，而订阅者也可能存在多个，它们都能接收到相同主题的消息。生活中的报纸订阅就是一种典型的发布 / 订阅模型。

比较酷的是 Kafka 同时支持这两种消息引擎模型，专栏后面我会分享 Kafka 是如何做到这一点的。

提到消息引擎系统，你可能会问 JMS 和它是什么关系。JMS 是 Java Message Service，它也是支持上面这两种消息引擎模型的。严格来说它并非传输协议而仅仅是一组 API 罢了。不过可能是 JMS 太有名气以至于很多主流消息引擎系统都支持 JMS 规范，比如 ActiveMQ、RabbitMQ、IBM 的 WebSphere MQ 和 Apache Kafka。当然 Kafka 并未完全遵照 JMS 规范，相反，它另辟蹊径，探索出了一条特有的道路。

好了，目前我们仅仅是了解了消息引擎系统是做什么的以及怎么做的，但还有个重要的问题是为什么要使用它。

依旧拿上面“民间版”举例，我们不禁要问，为什么系统 A 不能直接发送消息给系统 B，中间还要隔一个消息引擎呢?

答案就是“**削峰填谷**”。这四个字简直比消息引擎本身还要有名气。

我翻了很多文献，最常见的就是这四个字。所谓的“削峰填谷”就是指缓冲上下游瞬时突发流量，使其更平滑。特别是对于那种发送能力很强的上游系统，如果没有消息引擎的保护，“脆弱”的下游系统可能会直接被压垮导致全链路服务“雪崩”。但是，一旦有了消息引擎，它能够有效地对抗上游的流量冲击，真正做到将上游的“峰”填满到“谷”中，避免了流量的震荡。消息引擎系统的另一大好处在于发送方和接收方的松耦合，这也在一定程度上简化了应用的开发，减少了系统间不必要的交互。

说了这么多，可能你对“削峰填谷”并没有太多直观的感受。我还是举个例子来说明一下 Kafka 在这中间是怎么去“抗”峰值流量的吧。回想一下你在极客时间是如何购买这个课程的。如果我没记错的话极客时间每门课程都有一个专门的订阅按钮，点击之后进入到付费页面。这个简单的流程中就可能包含多个子服务，比如点击订阅按钮会调用订单系统生成对应的订单，而处理该订单会依次调用下游的多个子系统服务 ，比如调用支付宝和微信支付的接口、查询你的登录信息、验证课程信息等。显然上游的订单操作比较简单，它的 TPS 要远高于处理订单的下游服务，因此如果上下游系统直接对接，势必会出现下游服务无法及时处理上游订单从而造成订单堆积的情形。特别是当出现类似于秒杀这样的业务时，上游订单流量会瞬时增加，可能出现的结果就是直接压跨下游子系统服务。

解决此问题的一个常见做法是我们对上游系统进行限速，但这种做法对上游系统而言显然是不合理的，毕竟问题并不出现在它那里。所以更常见的办法是引入像 Kafka 这样的消息引擎系统来对抗这种上下游系统 TPS 的错配以及瞬时峰值流量。

还是这个例子，当引入了 Kafka 之后。上游订单服务不再直接与下游子服务进行交互。当新订单生成后它仅仅是向 Kafka Broker 发送一条订单消息即可。类似地，下游的各个子服务订阅 Kafka 中的对应主题，并实时从该主题的各自分区（Partition）中获取到订单消息进行处理，从而实现了上游订单服务与下游订单处理服务的解耦。这样当出现秒杀业务时，Kafka 能够将瞬时增加的订单流量全部以消息形式保存在对应的主题中，既不影响上游服务的 TPS，

同时也给下游子服务留出了充足的时间去消费它们。这就是 Kafka 这类消息引擎系统的最大意义所在。

如果你对 Kafka Broker、主题和分区等术语还不甚了解的话也不必担心，我会在专栏后面专门花时间介绍一下 Kafka 的常见概念和术语。

在今天结束之前，我还想和你分享一个自己的小故事。在 2015 年那会儿，我花了将近 1 年的时间阅读 Kafka 源代码，期间多次想要放弃。你要知道阅读将近 50 万行源码是多么痛的领悟。我还记得当初为了手写源代码注释，自己写满了一个厚厚的笔记本。不过幸运的是我坚持了下来，之前的所有努力也没有白费，以至于后面写书、写极客时间专栏就变成了一件件水到渠成的事情。

最后我想送给你一句话：**聪明人也要下死功夫**。我不记得这是曾国藩说的还是季羡林说的，但这句话对我有很大影响，当我感到浮躁的时候它能帮我静下心来踏踏实实做事情。希望这句话对你也有所启发。切记：聪明人要下死功夫！

消息引擎系统ABC

- Apache Kafka是一款开源的消息引擎系统。根据维基百科的定义，消息引擎系统是一组规范。企业利用这组规范在不同系统之间传递语义准确的消息，实现松耦合的异步式数据传递。通俗来讲，就是系统A发送消息给消息引擎系统，系统B从消息引擎系统中读取A发送的消息。

- 消息引擎系统要设定具体的传输协议，即我用什么方法把消息传输出去，常见的方法有2种：点对点模型；发布/订阅模型。Kafka同时支持这两种消息引擎模型。

- 系统A不能直接发送消息给系统B，中间还要隔一个消息引擎呢，是为了“削峰填谷”。

开放讨论

请谈谈你对消息引擎系统的理解，或者分享一下你的公司或组织是怎么使用消息引擎来处理实际问题的。

欢迎写下你的思考和答案，我们一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。

AI智能总结

Apache Kafka: 解耦异步数据传递的利器

Apache Kafka是一款开源的消息引擎系统，旨在实现系统间的松耦合异步数据传递。它通过消息队列和发布/订阅模型传输消息，同时支持JMS规范，主要用于“削峰填谷”，即缓冲上下游瞬时突发流量，使其更平滑，避免流量的震荡。Kafka使用纯二进制的字节序列作为消息格式，并具有点对点模型和发布/订阅模型的双重支持。引入Kafka后，上游系统不再直接与下游子服务进行交互，而是通过Kafka Broker发送和接收消息，实现了上游和下游服务的解耦，从而有效应对瞬时峰值流量。作者分享了对Kafka源代码的阅读经历，强调了“聪明人也要下死功夫”的重要性。这篇文章深入浅出地介绍了Kafka的特点和应用，对于想要快速了解消息引擎系统的读者来说，是一份有价值的技术概览。

全部留言 (121)

最新　精选

孙志强
2019-06-04

讲讲怎么把50完行源代码读下来的？嘿嘿

作者回复: 一行一行啃下来的。如果你也有兴趣，我建议可以先从kafka.log包开始读起，会很有收获的~~

共 6 条评论>　　68

开发无止境，BUG随身...
2019-06-03

有个问题请教下老师:
之前也用过kafka，怎么解决实时结果响应问题呢？比如秒杀商品，生产者产生订单，消费者处理订单结果，那这结果如何实时返回给用户呢？

作者回复: 这个场景使用Kafka Streams比较适合，它就是为read-process-write场景服务的

共 2 条评论> 60

Lei Yang
2019-06-03

老师可以讲一讲Kafka和别的mq的区别和最佳选择方法么？例如什么时候选择RabbitMQ什么时候选择Kafka等等

作者回复: RabbitMQ属于比较传统的消息队列系统，支持标准的消息队列协议（AMQP, STOMP，MQTT等），如果你的应用程序需要支持这些协议，那么还是使用RabbitMQ。另外RabbitMQ支持比较复杂的consumer Routing，这点也是Kafka不提供的。

共 2 条评论> 52

Dovelol
2019-06-03

老师好，想问下有些业务用mq来做异步处理，为了削峰填谷，是不是上游发送消息成功就认为业务成功了，可能下游过很久去消费，那实时性要求很高的业务怎么办呢，比如生成了订单但是一直不处理也不好吧。另外想请教下老师的角度来讲下mq和rpc调用的区别是什么呢?

作者回复: mq和rpc的区别往大了说属于数据流模式（dataflow mode）的问题。我们常见的数据流有三种：1. 通过数据库；2. 通过服务调用（REST/RPC）；3. 通过异步消息传递（消息引擎，如Kafka）

RPC和MQ是有相似之处的，毕竟我们远程调用一个服务也可以看做是一个事件，但不同之处在于：

1. MQ有自己的buffer，能够对抗过载（overloaded）和不可用场景
2. MQ支持重试
3. 允许发布/订阅模式

当然它们还有其他区别。应该这样说RPC是介于通过数据库和通过MQ之间的数据流模式。

共 5 条评论> 51

Shane
2019-06-14

老师，今天才学习到这篇文章，还是老师能够在百忙之中抽出时间来解答我的困惑。
这篇文章提到了消息的协议，老师这里介绍了两种模式一种是点对点，一种是订阅，发布模式。但是，为什么我一开始想到消息的协议是http之类的传输协议？这两个有什么区别和联系?

作者回复: http不属于消息传输协议，它是网络通信协议的一种，严格来说这是两个范畴或者说是两个层次上的协议。

通常来说，两个进程进行数据流交互的方式一般有三种：
1. 通过数据库：进程1写入数据库；进程2读取数据库
2. 通过服务调用：比如REST或RPC，而HTTP协议通常就作为REST方式的底层通讯协议
3. 通过消息传递的方式：进程1发送消息给名为broker的中间件，然后进程2从该broker中读取消息。消息传输协议属于这种模式

因此我说虽然我们都称它们为协议，但它们不是一个层次上的协议。

共 4 条评论>

39

jeffery
2019-06-03

pulsar高吞吐低延迟和kafka谁会主宰未来？夕哥、能不能拓展下flink+kafka的耦合！谢谢

作者回复: 和Pulsar的斯杰、翟佳都相识，不敢妄下结论。Flink + Kafka最近的确有标准套餐的趋势：）

共 2 条评论>

27

杨鹏程baci
2019-06-20

胡夕老师好，我是第一次在这提问，这门课程我应该是0基础了，有一些疑问希望老师帮忙解答一下，用消息引擎的这种数据流数据方式，上游是不是就无法得知处理结果了，甚至是无法将返回值传回上游了？谢谢！

作者回复: 嗯嗯，确实不太容易。因为这种通信方式一般是异步且是单向的，如果你需要这种回馈机制，最好使用服务调用 的方式

共 3 条评论>

23

gnayuh
2019-06-03

老师讲的很好，我是之前那种听过kafaka等消息引擎大名的初学者，听完第一节课，联想到这个发布订阅模型跟之前学过java设计模式之观察者，总感觉它们之间有那么点类似，想知道

它们之间的某种关系。还有就是学操作系统时候的生产者消费者也很像

作者回复: 它们的确很类似。特别是发布/订阅与观察者模式。在《Head first Design Pattern》一书中更是有这样的话： Publishers + Subscribers = Observer Pattern

不过细究起来还是有些许不同，pub与sub之间通常都隔了一层，比如broker或message channel，但是Observer模式中Observer通常都直接对接被观测者，因此Pub/Sub模式中组件的耦合度更低；另外Pub/Sub经常是以异步的方式实现，而Observer模式通常都是同步的

22

燕子上
2019-06-04

我司直接把kafka当mq来使用，高吞吐、低延迟、松耦合。对！我司看上了松耦合，哪哪都要用kafka解耦，真正的面向kafka编程🤣

作者回复: 赞👍

共 2 条评论>

21

Bin滨
2019-06-19

谢谢知识分享。
在Martin Kleppmann 的书中把kafka 定义成 log-based message brocker， 这个基本上是对kafka最简单的定义了。append-only， partition， totally ordered 是比较需要理解的概念。

作者回复: 嗯嗯，DDIA是一本神书：）

其实这个定义最早还是Kafka作者Jay Kreps提出的，有兴趣可以看看Kafka的论文：http://notes.stephenholiday.com/Kafka.pdf
以及Jay Kreps的 《I ❤️ Logs》

17