03-消息模型：主题和队列有什么区别？

你好，我是李玥。这节课我们来学习消息队列中像队列、主题、分区等基础概念。这些基础的概念，就像我们学习一门编程语言中的基础语法一样，你只有搞清楚它们，才能进行后续的学习。

如果你研究过超过一种消息队列产品，你可能已经发现，每种消息队列都有自己的一套消息模型，像队列（Queue）、主题（Topic）或是分区（Partition）这些名词概念，在每个消息队列模型中都会涉及一些，含义还不太一样。

为什么出现这种情况呢？因为没有标准。曾经，也是有一些国际组织尝试制定过消息相关的标准，比如早期的JMS和AMQP。但让人无奈的是，标准的进化跟不上消息队列的演进速度，这些标准实际上已经被废弃了。

那么，到底什么是队列？什么是主题？主题和队列又有什么区别呢？想要彻底理解这些，我们需要从消息队列的演进说起。

主题和队列有什么区别？

在互联网的架构师圈儿中间，流传着这样一句不知道出处的名言，我非常认同和喜欢：好的架构不是设计出来的，而是演进出来的。 现代的消息队列呈现出的模式，一样是经过之前的十几年逐步演进而来的。

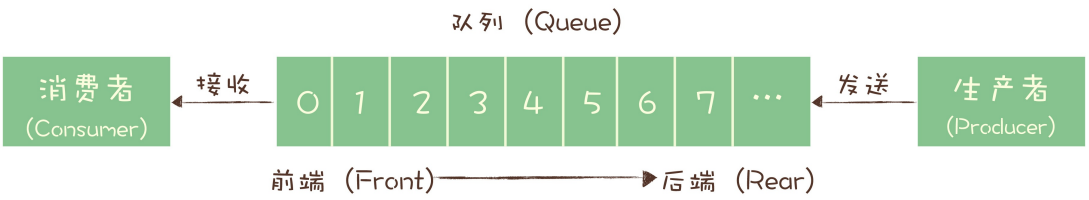
最初的消息队列，就是一个严格意义上的队列。在计算机领域，“队列（Queue）”是一种数据结构，有完整而严格的定义。在维基百科中，队列的定义是这样的：

> 队列是先进先出（FIFO, First-In-First-Out）的线性表（Linear List）。在具体应用中通常用链表或者数组来实现。队列只允许在后端（称为rear）进行插入操作，在前端（称为front）进行删除操作。

这个定义里面包含几个关键点，第一个是先进先出，这里面隐含着的一个要求是，在消息入队出队过程中，需要保证这些消息**严格有序**，按照什么顺序写进队列，必须按照同样的顺序从队列中读出来。不过，队列是没有“读”这个操作的，“读”就是出队，也就是从队列中“删除”这条消息。

早期的消息队列，就是按照“队列”的数据结构来设计的。我们一起看下这个图，生产者（Producer）发消息就是入队操作，消费者（Consumer）收消息就是出队也就是删除操作，服务端存放消息的容器自然就称为“队列”。

这就是最初的一种消息模型：队列模型。



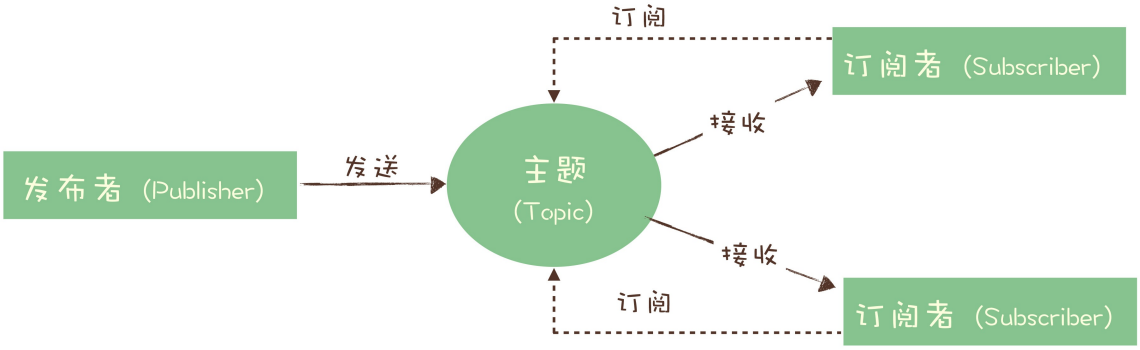
如果有多个生产者往同一个队列里面发送消息，这个队列中可以消费到的消息，就是这些生产者生产的所有消息的合集。消息的顺序就是这些生产者发送消息的自然顺序。如果有多个消费者接收同一个队列的消息，

这些消费者之间实际上是竞争的关系，每个消费者只能收到队列中的一部分消息，也就是说任何一条消息只能被其中的一个消费者收到。

如果需要将一份消息数据分发给多个消费者，要求每个消费者都能收到全量的消息，例如，对于一份订单数据，风控系统、分析系统、支付系统等都需要接收消息。这个时候，单个队列就满足不了需求，一个可行的解决方式是，为每个消费者创建一个单独的队列，让生产者发送多份。

显然这是个比较蠢的做法，同样的一份消息数据被复制到多个队列中会浪费资源，更重要的是，生产者必须知道有多少个消费者。为每个消费者单独发送一份消息，这实际上违背了消息队列“解耦”这个设计初衷。

为了解决这个问题，演化出了另外一种消息模型：“**发布-订阅模型（Publish-Subscribe Pattern）**”。



在发布-订阅模型中，消息的发送方称为发布者（Publisher），消息的接收方称为订阅者（Subscriber），服务端存放消息的容器称为主题（Topic）。发布者将消息发送到主题中，订阅者在接收消息之前需要先“订阅主题”。“订阅”在这里既是一个动作，同时还可以认为是主题在消费时的一个逻辑副本，每份订阅中，订阅者都可以接收到主题的所有消息。

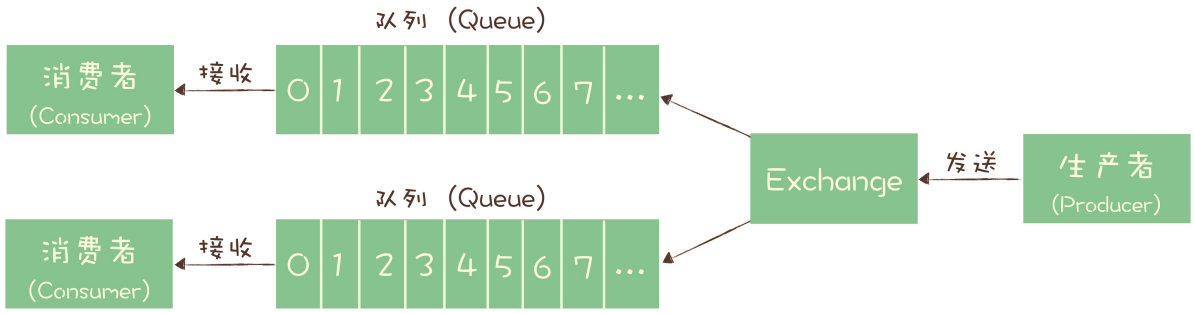
在消息领域的历史上很长的一段时间，队列模式和发布-订阅模式是并存的，有些消息队列同时支持这两种消息模型，比如ActiveMQ。我们仔细对比一下这两种模型，生产者就是发布者，消费者就是订阅者，队列就是主题，并没有本质的区别。**它们最大的区别其实就是，一份消息数据能不能被消费多次的问题。**

实际上，在这种发布-订阅模型中，如果只有一个订阅者，那它和队列模型就基本是一样的了。也就是说，发布-订阅模型在功能层面上是可以兼容队列模型的。

现代的消息队列产品使用的消息模型大多是这种发布-订阅模型，当然也有例外。

RabbitMQ的消息模型

这个例外就是RabbitMQ，它是少数依然坚持使用队列模型的产品之一。那它是怎么解决多个消费者的问题呢？你还记得我在上节课中讲到RabbitMQ的一个特色Exchange模块吗？在RabbitMQ中，Exchange位于生产者和队列之间，生产者并不关心将消息发送给哪个队列，而是将消息发送给Exchange，由Exchange上配置的策略来决定将消息投递到哪些队列中。



同一份消息如果需要被多个消费者来消费，需要配置Exchange将消息发送到多个队列，每个队列中都存放一份完整的消息数据，可以为一个消费者提供消费服务。这也可以变相地实现新发布-订阅模型中，“一份消息数据可以被多个订阅者来多次消费”这样的功能。具体的配置你可以参考RabbitMQ官方教程，其中一个章节专门是讲如何实现发布订阅的。

RocketMQ的消息模型

讲完了RabbitMQ的消息模型，我们再来看看RocketMQ。RocketMQ使用的消息模型是标准的发布-订阅模型，在RocketMQ的术语表中，生产者、消费者和主题与我在上面讲的发布-订阅模型中的概念是完全一样的。

但是，在RocketMQ也有队列（Queue）这个概念，并且队列在RocketMQ中是一个非常重要的概念，那队列在RocketMQ中的作用是什么呢？这就要从消息队列的消费机制说起。

几乎所有的消息队列产品都使用一种非常朴素的“请求-确认”机制，确保消息不会在传递过程中由于网络或服务器故障丢失。具体的做法也非常简单。在生产端，生产者先将消息发送给服务端，也就是Broker，服务端在收到消息并将消息写入主题或者队列中后，会给生产者发送确认的响应。

如果生产者没有收到服务端的确认或者收到失败的响应，则会重新发送消息；在消费端，消费者在收到消息并完成自己的消费业务逻辑（比如，将数据保存到数据库中）后，也会给服务端发送消费成功的确认，服务端只有收到消费确认后，才认为一条消息被成功消费，否则它会给消费者重新发送这条消息，直到收到对应的消费成功确认。

这个确认机制很好地保证了消息传递过程中的可靠性，但是，引入这个机制在消费端带来了一个不小的问题。什么问题呢？为了确保消息的有序性，在某一条消息被成功消费之前，下一条消息是不能被消费的，否则就会出现消息空洞，违背了有序性这个原则。

也就是说，每个主题在任意时刻，至多只能有一个消费者实例在进行消费，那就没法通过水平扩展消费者的数量来提升消费端总体的消费性能。为了解决这个问题，RocketMQ在主题下面增加了队列的概念。

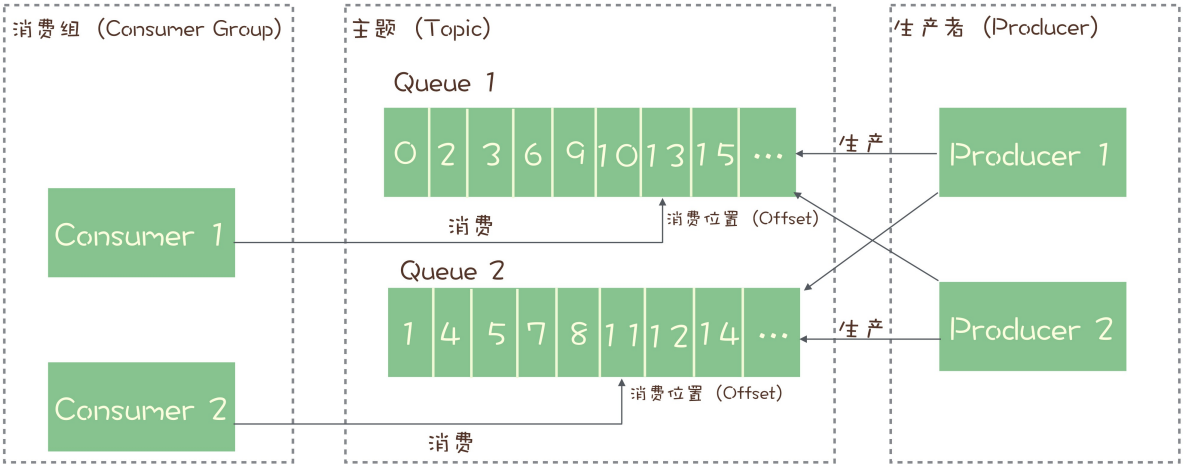
每个主题包含多个队列，通过多个队列来实现多实例并行生产和消费。需要注意的是，RocketMQ只在队列上保证消息的有序性，主题层面是无法保证消息的严格顺序的。

RocketMQ中，订阅者的概念是通过消费组（Consumer Group）来体现的。每个消费组都消费主题中一份完整的消息，不同消费组之间消费进度彼此不受影响，也就是说，一条消息被Consumer Group1消费过，也会再给Consumer Group2消费。

消费组中包含多个消费者，同一个组内的消费者是竞争消费的关系，每个消费者负责消费组内的一部分消息。如果一条消息被消费者Consumer1消费了，那同组的其他消费者就不会再收到这条消息。

在Topic的消费过程中，由于消息需要被不同的组进行多次消费，所以消费完的消息并不会立即被删除，这就需要RocketMQ为每个消费组在每个队列上维护一个消费位置（Consumer Offset），这个位置之前的消息都被消费过，之后的消息都没有被消费过，每成功消费一条消息，消费位置就加一。这个消费位置是非常重要的概念，我们在使用消息队列的时候，丢消息的原因大多是由于消费位置处理不当导致的。

RocketMQ的消息模型中，比较关键的概念就是这些了。为了便于你理解，我画了下面这张图：



你可以对照这张图再把我刚刚讲的这些概念继续消化一下，加深理解。

Kafka的消息模型

我们再来看看另一种常见的消息队列Kafka，Kafka的消息模型和RocketMQ是完全一样的，我刚刚讲的所有RocketMQ中对应的概念，和生产消费过程中的确认机制，都完全适用于Kafka。唯一的区别是，在Kafka中，队列这个概念的名称不一样，Kafka中对应的名称是“分区（Partition）”，含义和功能是没有任何区别的。

小结

我们来总结一下本节课学习的内容。首先我们讲了队列和主题的区别，这两个概念的背后实际上对应着两种不同的消息模型：队列模型和发布-订阅模型。然后你需要理解，这两种消息模型其实并没有本质上的区别，都可以通过一些扩展或者变化来互相替代。

常用的消息队列中，RabbitMQ采用的是队列模型，但是它一样可以实现发布-订阅的功能。RocketMQ和Kafka采用的是发布-订阅模型，并且二者的消息模型是基本一致的。

最后提醒你一点，我这节课讲的消息模型和相关的概念是业务层面的模型，深刻理解业务模型有助于你用最佳的姿势去使用消息队列。

但业务模型不等于就是实现层面的模型。比如说MySQL和Hbase同样是支持SQL的数据库，它们的业务模型中，存放数据的单元都是“表”，但是在实现层面，没有哪个数据库是以二维表的方式去存储数据的，MySQL使用B+树来存储数据，而HBase使用的是KV的结构来存储。同样，像Kafka和RocketMQ的业务模型基本是一样的，并不是说他们的实现就是一样的，实际上这两个消息队列的实现是完全不同的。

思考题

最后给大家留一个思考题。刚刚我在介绍RocketMQ的消息模型时讲过，在消费的时候，为了保证消息的不丢失和严格顺序，每个队列只能串行消费，无法做到并发，否则会出现消费空洞的问题。那如果放宽一下限制，不要求严格顺序，能否做到单个队列的并行消费呢？如果可以，该如何实现？欢迎在留言区与我分享讨论。

感谢阅读，如果你觉得这篇文章对你有一些启发，也欢迎把它分享给你的朋友。

精选留言：

- a、 2019-07-27 12:36:34
  今天的思考题，我觉得应该是，把消息队列的先进先出，改成数组的随机访问，用offset来控制消息组具体要消费哪条消息，mq不主动删除消息，消息有过期时间，如果到了过期时间，只能确认不能重新该消费，只保留最大可设置天数的消息。超过该天数则删除，还要维护客户端确认信息，如果有客户端没确认，需要有重发机制。不知道这个想法对不对？ [10赞]

  作者回复2019-07-27 13:32:20
  现代的消息队列大多就是这么实现的。

- 发条橙子 。 2019-07-27 14:43:36
  老师， 初学者有一些疑问的地方 ，希望老师可以帮忙解答 😁

  Rocket mq的模型图有些没有看懂， 模拟下场景；比如生产者是商品中心发送商品状态的更新（主题）消息（比如下架），那商品中心有多台机器就对应多个producer 。消费者组有两个，分别是导购中心 和活动中心。

  疑问一 ：图中主题的队列是有多少个消费者组就有多少个队列么，是根据我们配置的消费者组数，mq自动增加主题队列个数么

  疑问二 ：看到图中每个producer的消息都往所有的队列中添加一条消息，每个消费者组消费自己的队列，但没有看懂这个队列是如何支持 “当水平扩展消费者机器” 可以加快消息的处理 。 每个消费组应该都

是按照队列等前一条处理完了，才能去处理下一条（ps：这样来看，rb mq也是这个样子，如何通过水平扩展机器来加快消息的处理）

疑问三： 图中队列的消费位置有点没看懂，我看是全剧唯一的，这是为什么，每个队列不能都是从0到1么 [6赞]

作者回复2019-07-27 21:44:12
A1： 不是，消费者组和队列数没有关系，你这个例子中消费者组的数量是2个。队列数量可以根据数据量和消费速度来合理配置。RocketMQ和Kafka都可以支持水平扩容队列数量，但是都需要手动操作。

A2：producer会往所有队列发消息，但不是“同一条消息每个队列都发一次”，每条消息只会往某个队列里面发送一次。

对于一个消费组，每个队列上只能串行消费，多个队列加一起就是并行消费了，并行度就是队列数量，队列数量越多并行度越大，所以水平扩展可以提升消费性能。

A3：每队列每消费组维护一个消费位置（offset），记录这个消费组在这个队列上消费到哪儿了。

- Geek_de6f9a 2019-07-28 21:57:16
老师你好，想请问一下消费的顺序问题。
对于有的消息，需要保证顺序性，比如交易状态和im消息。像im消息还要保证唯一性。

Q1: rocketmq，一个消费组在一个主题下的多个队列并发消费就无法保证消息的顺序性。这种该如何处理?

Q2: 客户端和mq要保持一种重试的机制，如果在网络延迟出现问题的时候，前面的消息一直未收到ack响应，若不做任何处理，后面的就会阻塞，还是重试之后放弃，若是不能发生丢失的信息该如何处理。

Q3: 如何保证消息的唯一性，在重试的过程中，第一条消息已经发送，未收到ack，则进行第二次重试。此时网络故障恢复，则客户端会收到两条消息，客户端如何保证消息的唯一性。 [4赞]

作者回复2019-07-29 10:12:43
A1：按照订单ID或者用户ID，用一致性哈希算法，计算出队列ID，指定队列ID发送，这样可以保证相同的订单/用户的消息总被发送到同一个队列上，就可以确保严格顺序了。

A2：会有一个超时，超时之前会阻塞，超时之后就解除锁定，允许其他消费者来拉消息，由于消费位置没变，下次再有消费者来这个队列拉消息，返回的还是上一条消息。

A3：这个问题我在后面的课中会专门来讲。

- DesertSnow 2019-07-27 09:37:11
没啥问题，就是想点个赞，老师的声音很nice！ [4赞]

作者回复2019-07-27 14:29:16
谢谢

- 川杰 2019-07-27 13:00:29
老师你好，RocketMQ中，消费位置(5)记录了当前消费组GroupA在A队列中的消费位置，(5)之前都被消费过，(5)之后都没有；那么这个(5)最终的作用是什么？
是当GroupA再取下一个消息时，用来判断在队列A中的消息位置用的吗？除此之外还有其他作用吗？ [3赞]

作者回复2019-07-27 14:41:09
就是记录哪些消息消费了，哪些没消费。由于消费者是不记录消费位置的，它消费的时候只管去找Broker要消息，Broker必须知道消费到哪儿了，好找出下一条或下一批消息给客户端。

- ly 2019-07-27 12:16:31
老师您好，关于rocketmq的那张图有几个疑问：
consumergourp中的某个consumer是和某个具体的queue一一关联绑定的么？还是说某consumer每次都随机从某queue消费，另外如果是一一关联的话，那某个consumer挂了，那关联的那个queue的消息该由哪个consumer来接替消费呢？
另外product发给topic的消息是否是被topic随机分配到某个queue中的？还是说product必须指定发到哪个queue中？ [3赞]

作者回复2019-07-27 13:43:29
第一个问题，consumer和queue不是强关联的，但是在任何一个时刻，某个queue在同一个consumer group中最多只能有一个consumer占用。

第二个问题，producer和queue不需要关联，简单点儿说，就是发到哪个queue都可以。RocketMQ的默认策略是轮询选择每个queue。

- flyamonkey 2019-07-27 11:39:01
不要求严格顺序的话，应该是可以做到单个队列并行的，但这种情况下消息的消费可能就是个出队操作，而非等待消费端的ack后再出队了，这样势必会造成消息的丢失，所以需要有一定的补偿机制，如消息的重传和持久化等。个人见解，不知道是不是准确，还请老师指点~ [3赞]

作者回复2019-07-27 13:46:59
没错！具体可以看一下RocketMQ的并行消费的实现。

- 书策稠浊 2019-07-27 10:32:13
Rocket mq那张图是不是有问题，consumer是不直接对topic的，group才直接对topic，求解答，谢谢。 [3赞]

作者回复2019-07-27 14:38:03
consumer在某个时刻对应的是某个queue（图中的实线），consumer group 对应 topic（同样是虚线方框），我理解这张图和你的描述是一致的。

- Penn 2019-07-27 09:15:41
维护一个offset抽象，offset由单个位置变成一个集合，集合中包含多个单个位置。类似多值信号量的机制 [2赞]

作者回复2019-07-27 14:29:00
✍

- 苏志辉 2019-07-28 17:00:01
要保证严格顺序，必须指定这个顺序的消息在同一个队列中，也就是必须保险这些消息路由到的队列是一个值吧？否则同一主题，多个队列的原因没法保证顺序，只是单队列纬度有序 [1赞]

作者回复2019-07-29 10:07:31
没错。

- Geek_87338d 2019-07-28 15:25:47

有三个问题没太想清楚，希望老师解答一下。
1. rocket mq的模型，是不是每有一个新的consumer，都需要对mq进行配置新增一个queue？（我预设了一个前提是1queue有且只有1consumer来消费，1consumer只消费1queue不知道对不对）这样下游机器重启或者加机器，运维要累死。但没想明白它是怎么解决新增或者减少consumer的问题的？
2. rocket mq的那个流程图，不能保证消息在全局顺序处理（比如处理0号消息的consumer1可能比处理1号消息的consumer2要慢，对于整个系统，1号消息被先处理），那么保证单个queue顺序处理的意义或者场景是什么呢？好像是为了消息的ack机制？
3. 每个消息都确认（tcp是发送方一直发，接收方只确认最后的sequence，这样快得多）效率很低，那是怎么做到打满网卡的？靠大量的queue并发吗？ [1赞]

作者回复2019-07-29 10:05:00
A1：队列只有一份，无论有多少订阅，所以不存在你说的问题。
A2：目前的这种设计也是没办法的办法，还没有什么完美的解决方案既在topic上保证严格顺序，又要保证高性能和数据可靠性。但是目前这种实现也可以解决很多对顺序有要求的场景的问题。

A3：实际上并不是一条一条确认的，而是一批一批确认的。一般consumer取一批消息，然后确认的时候直接提交这批消息中最后一条消息的位置来确认这批消息。

- 微微一笑 2019-07-28 12:15:47
课后练习：同一个消费组，如果在一个消息队列里不保证严格顺序的情况下，实现并行消费，我觉得关键点在于这个队列的offset的分配。假设此时offset = m，为消费者a分配m位点，m+ = 1（注意：1，这个操作必须保证原子性；2，不用等待上个消息是否成功消费此消息）。然后为消费者b分配m位点，m+ = 1，以此类推，达到并行消费。（为保证消息的不丢失，需要为每条消息设置一个状态，标记是否被某个消费组成功消费，若消费失败，需要另起任务来做重试的工作） [1赞]

作者回复2019-07-29 09:55:57
可行。

- Mark Yao 2019-07-28 01:15:59
我觉得可以实现。我的思路：

1，题目中提到并发，想到多线程，但kafka的消费端不是线程安全的，不支持直接多线程消费

2，把消息放到类似管道队列后立刻确认消息，之后多线程处理数据。Java并发包有多个并发队列其中SynchronousQueue是一个不存储元素的阻塞队列。每一个put操作必须等待一个take操作，否则不能继续添加元素。队列本身不存储任何元素，吞吐量非常高。

3，单独启动了一个线程读取阻塞队列数据，然后放入线程池进行处理

4，2中提前确认了消息会导致处理消息的应用挂了消息丢失。解决这个问题可以在确认消息之前写库或记日志或者放在redis，这样保证消息的消费可靠性。 [1赞]

作者回复2019-07-29 09:52:18
嗯，这是一种解决思路。

- QQ怪 2019-07-27 16:03:03
学习基础很重要，打卡 [1赞]

- AgCl 2019-07-27 10:28:21
不同队列某种程度实现的是并行，一个队列的消费实现并行 [1赞]

- 一夫 2019-07-27 07:42:15
  老师，是目前所有的消息队列都是只能接受一个消费者消费完成后才能给下一个消费者吗？这样还有多消费者的意义吗？或者说是通过其他手段，比如获取后直接标记消费成功，如果处理失败，重新放回队列尾部这类的思路来实现并发多消费者呢？ [1赞]

  作者回复2019-07-27 14:22:56
  第一个问题，据我所知，大部分消息队列都是这样的：在任何一个时刻，某个队列，在一个消费组（Consumer Group ）内，只能有一个消费者占用。

  这么设计是为了保证严格顺序。

  所以在需要严格顺序的场景下，一般都建议消费者的数量和队列(kafka叫分区）保持一致。

  如果不要求严格顺序的话，单个队列上，是可以多个消费者并行消费的。

- Black 2019-07-30 00:14:02
  老师你好，Redis也有队列和pub/sub模型，能和这些mq一起对比一下嘛

- Black King Bar 2019-07-29 22:22:07
  读写分离就可以并行了吧，读并行不会消费空洞，写串行。

- DAV 2019-07-29 20:49:25
  两点拙见：
  1. RabbitMQ的FANOUT模式可以支持一天消息多次消费
  2. 并行消费队列如果不修改源码，可以在消费端再加一层控制逻辑，比如连续读取10条分给10个线程去处理，这样就近似并行消费了，原理就是加一个预消费层

- 二星球 2019-07-29 20:25:04
  老师好，请教个问题，消费端向服务端拉10条数据，offset偏移10个位置，如果消费端正确处理了其中的9条数据，其中一条数据异常，消息回滚，只能把10条数据都回滚，offset这种机制无法做到回滚某几条吧？如果业务处理一条数据很耗时，这是一种弊端吧，有没有好的解决办法呢