21 | 分布式通信之消息队列：货物自取

2019-11-11 聂鹏程 来自北京

《分布式技术原理与算法解析》

你好，我是聂鹏程。今天，我来继续带你打卡分布式核心技术。

在上一篇文章，我带你学习了分布式通信技术中的发布订阅。总结来说，发布订阅就是发布者产生数据到消息中心，订阅者订阅自己感兴趣的消息，消息中心根据订阅者的订阅情况，将相关消息或数据发送给对应的订阅者。所以，我将其思想，概括为“送货上门”。

在实际使用场景中，还有一种常用的通信方式，就是将消息或数据放到一个队列里，谁需要谁就去队列里面取。在分布式领域中，这种模式叫“消息队列”。与发布订阅相比，消息队列技术的核心思想可以概括为“货物自取”。

接下来，我们就一起打卡分布式通信技术中的消息队列吧。

什么是消息队列？

回想一下，在上一篇学术电子论文订阅的例子中，出版社或会议方将论文发布到论文网站（或平台）上，然后论文网站再将论文推送给订阅相关论文的老师或学生。这里的论文网站就是消息中心，负责根据订阅信息将论文送货上门，角色非常关键。

但其实，除了将论文送货上门外，我们还能想到另外一种模式，也就是出版社或会议方将论文发布到论文网站进行存储，老师或学生根据需要到论文网站按需购买文章。

这种思想，在分布式通信领域中称为消息队列模式，论文网站充当的就是消息队列的角色，也非常关键。接下来，**我再通过一个具体的应用案例来帮助你更加深入地理解什么是消息队列吧。**

比如，很多系统都提供了用户注册功能，注册完成后发送通知邮件。如下图所示，假设用户通过邮箱进行注册，填写完注册信息并点击提交后，系统的处理过程主要分为两步：

1. 检查用户注册信息的合法性，如果合法则将注册信息写入数据库中，若不合法，直接返回，流程结束；
2. 将用户注册信息写入数据库后，给用户发送通知邮件，以告知用户注册的相关信息，比如注册账号等信息。

假设，系统将注册信息写入数据库需要花费 400ms、给用户发送通知邮件需要花费 600ms。

这时，注册消息写入数据库和发送通知邮件这两个组件间是直接交互，且是同步通信方式。那么，从用户提交注册到收到响应，需要等系统完成这两个步骤。也就是说，如果不考虑通信延迟的话，注册系统对用户的响应时间是 1000ms，即 1s。

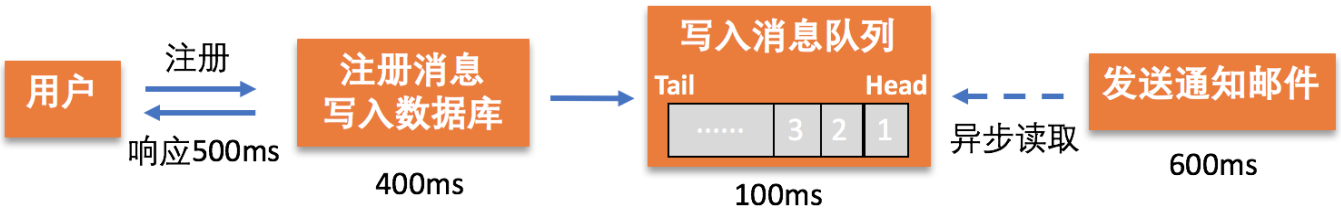
如下图所示，如果引入消息队列作为注册消息写入数据库和发送通知邮件这两个组件间的中间通信者，那么这两个组件就可以实现异步通信、异步执行。引入消息队列后，上述步骤可以分

为三步：

1. 检查用户注册信息的合法性，如果合法则将注册信息写入数据库中，若不合法则直接返回，流程结束；
2. 注册消息写入消息数据库后，将消息写入消息队列的队尾；
3. 发送通知邮件的组件去消息队列取出队首的消息，给用户发送通知邮件，告知用户注册的相关信息。

也就是说，采用消息队列模式，只需要第 2 步完成，即可给用户返回响应。第 3 步发送通知邮件可以在返回响应之后执行。

用户的注册信息写入数据库之后，通过数据库的可靠性设计来保证用户注册信息不会丢失，也就是说发送通知邮件的组件一定可以获取到用户注册信息，即保证会给注册用户发送通知邮件。也就是说，**消息队列的引入不会影响用户注册网站，但会提升用户响应效率。**



通常情况下，将消息写入消息队列的速度很快，假设需要 100ms。那么，引入消息队列后，发送通知邮件实现了异步读取，系统响应时间缩短为 500ms，响应速度提升了一倍，提升了用户体验。

讲完了用户注册这个例子，我们再来看消息队列的定义就比较容易理解了。

队列是一种具有先进先出特点的数据结构，**消息队列是基于队列实现的，存储具有特定格式的消息数据**，比如定义一个包含消息类型、标志消息唯一性的 ID、消息内容的一个结构体作为消息数据的特定格式。消息以特定格式放入这个队列的尾部后可以直接返回，并不需要系统马上处理，之后会有其他进程从队列头部开始读取消息，按照消息放入的顺序逐一处理。

从上面的例子中，我们也可以看出引入消息队列的好处是，提高响应速度，以及实现组件间的解耦。

消息队列的原理

现在，我把消息队列的工作原理从用户注册这个例子中剥离出来，给你一个更加直接的解释吧。

消息队列工作原理

消息队列的核心结构，如下图所示。与发布订阅模式类似，消息队列模式也是包括 3 个核心部分：

生产者。生产者会产生消息或数据，并将消息或数据插入到消息队列中。

消息队列。一种具有先进先出特点的数据结构，用于存储消息。

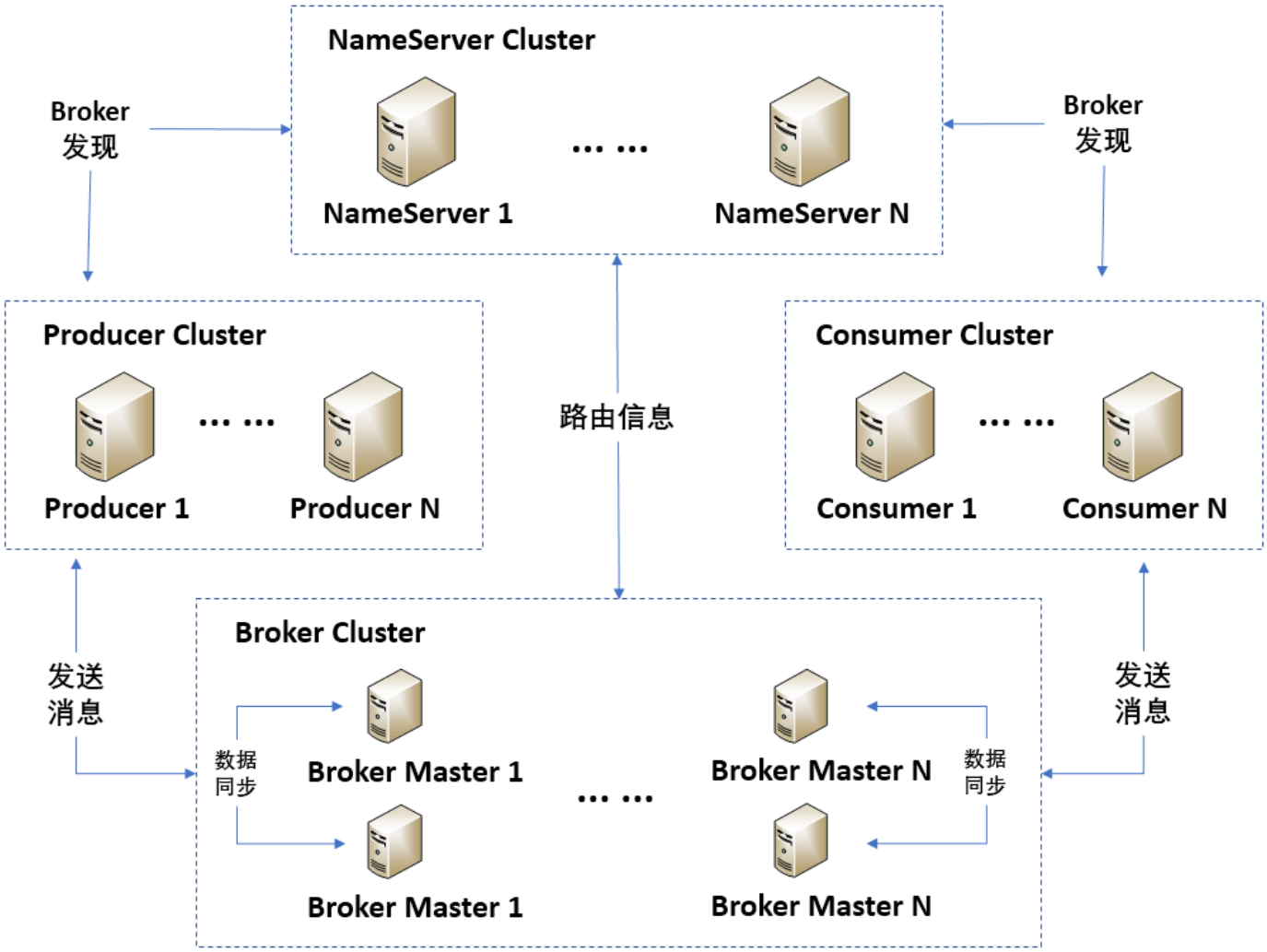
消费者。从消息队列中获取消息或数据，进行相关处理。

具体流程是，生产者将发送的消息插入消息队列，也就是入队，之后会有一个消费者从消息队列中逐次取出消息进行处理，完成出队。

了解了消息队列的工作原理，接下来我以阿里开源的 RocketMQ 为例，与你进一步介绍消息队列的原理、工作机制和实践应用。

RocketMQ 消息队列原理及工作机制

首先，我们看一下 RocketMQ 的架构图，形成一个整体认知。



RokcetMQ 共包括 NameServer Cluster、Producer Cluster、Broker Cluster 和 Consumer Cluster 共 4 部分。接下来，我们一起看看每部分的具体功能吧。

NameServer Cluster，指的是名字服务器集群。这个集群的功能与 Kafka 中引入的 ZooKeeper 类似，提供分布式服务的协同和管理功能，在 RocketMQ 中主要是管理 Broker 的信息，包括有哪些 Broker、Broker 的地址和状态等，以方便生产者获取 Broker 信息发布消息，以及订阅者根据 Broker 信息获取消息。

Producer Cluster，指的是生产者集群，负责接收用户数据，然后将数据发布到消息队列中心 Broker Cluster。那么，生产者按照集群的方式进行部署，好处是什么呢？在我看来，好处可以概括为以下两点：

一是，多个 Producer 可以并发接收用户的输入数据，提升业务处理效率；

二是，考虑到可靠性问题，如果只有一个 Producer 接收用户输入数据，当这个 Producer 故障后，整个业务就无法运行了。

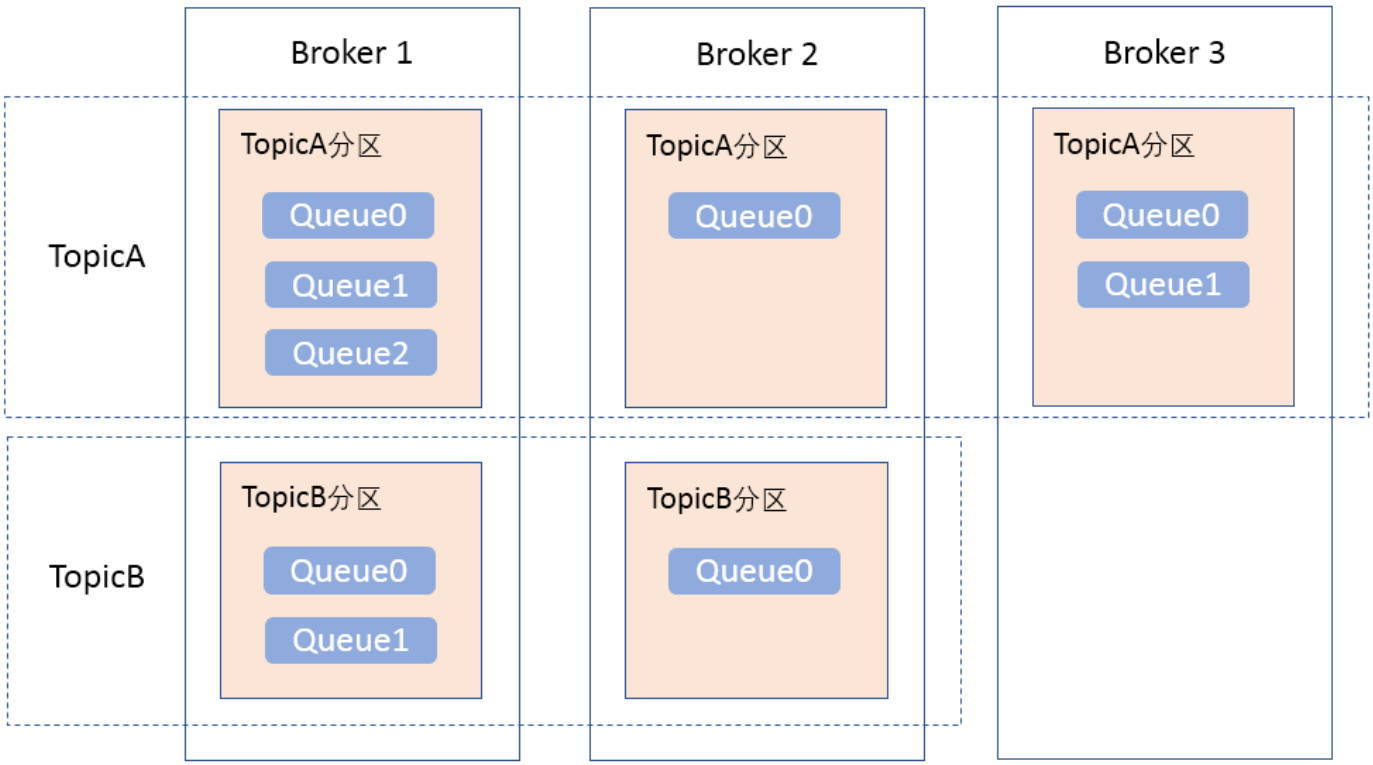
Consumer Cluster，指的是消费者集群，负责从 Broker 中获取消息进行消费。Consumer 以集群方式进行部署的好处是，提升消费者的消费能力，以避免消息队列中心存储溢出，消息被丢弃。

Broker Cluster，指的是 Broker 集群，负责存储 Producer Cluster 发布的数据，以方便消费者进行消费。

Broker Cluster 中的每个 Broker 都进行了主从设计，即每个 Broker 分为 Broker Master 和 Broker Slave，Master 既可以写又可以读，Slave 不可以写只可以读。每次 Broker Master 会把接收到的消息同步给 Broker Slave，以实现数据备份。一旦 Broker Master 崩溃了，就可以切换到 Broker Slave 继续提供服务。这种设计的好处是，提高了系统的可靠性。

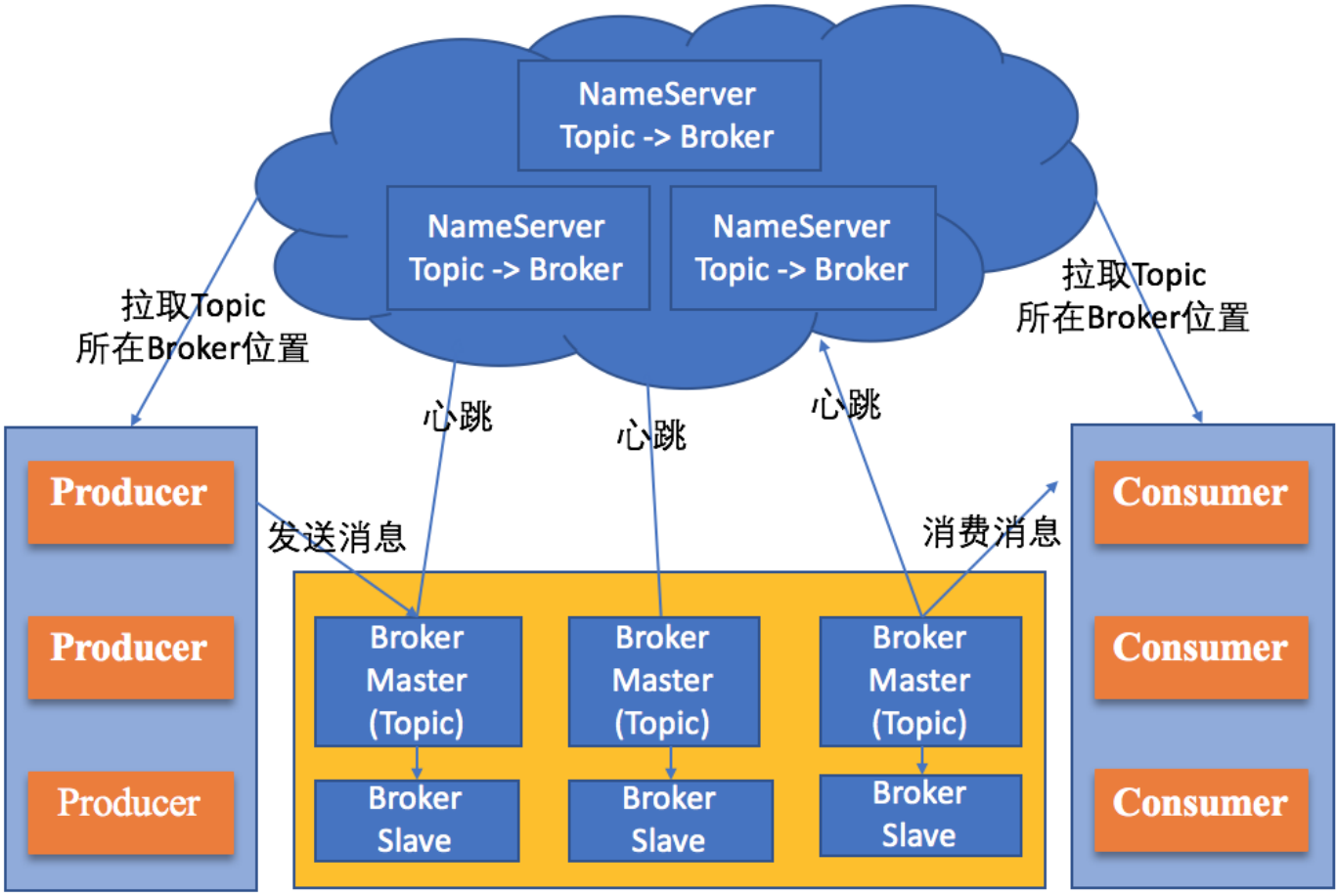
可以看出，Broker Cluster 就是我们今天要讲的核心“消息队列中心”，那么它到底是如何采用队列实现的呢？接下来，我们就一起看看 **Broker Cluster 的实现方式**吧。

如下图所示，在 Broker Cluster 中，消息的存储采用主题（Topic）+ 消息队列（Queue）的方式实现：



与 Kafka 一样，RocketMQ 中的主题也是一个逻辑概念。一个主题可以分区，分布在各个不同的 Broker 中，每个 Broker 上只有该主题的部分数据。每个主题分区中，队列的数量可以不同，由用户在创建主题时指定。队列是资源分配的基本单元，消息进行存储时会存放到相应主题的分区中。

上面我为你介绍了 RocketMQ 的关键组件。接下来，我们再看看 **RocketMQ 的工作流程**，如下图所示：



1. 首先启动 NameServer，然后启动 Broker。Broker 启动后，会主动找 NameServer 建立连接，并将自己的信息注册到 NameServer 上。注册完毕后，Broker 会周期性地给 NameServer 发送心跳包，比如每隔 1s 发送一次，以告知 NameServer 自己还活着；心跳包里还可以包括 Broker 当前存储的数据信息，也就是说 Broker 可以周期性地向 NameServer 更新自己的数据信息，以保证 NameServer 上存储的数据是最新的。
2. 创建主题，并确定这个主题的数据放入哪些 Broker 中。
3. 当 Producer 生产消息发送到主题时，需要先到 NameServer 查询该主题存放在哪些 Broker 中，获取到相关 Broker 信息后，将消息发送给这些 Broker 进行存储。
4. Consumer 要从主题消费消息，也需要首先到 NameServer 查询一下该主题的消息存储在哪些 Broker 上，然后去相应的 Broker 获取消息进行消费。

通过对 RocketMQ 的介绍，相信你已经对消息队列有比较深刻的认识了。接下来，我们再看看**消息队列模式适用于什么场景**吧。

消息队列模式，是根据消费者需求到消息队列获取数据消费的，消费者只需要知道消息队列地址即可，消息队列中心也无需提前知道消费者信息。也就是说，这种模式对消费者没有特别需求，因此比较适合消费者为临时用户的场景。

比如目前，阿里内部将 RocketMQ 应用于购物交易、充值、消息推送等多个场景，因为在这些场景下，每个消费者不是常驻进程或服务，几乎都是临时存在。此外，滴滴、联想等公司也都有采用 RocketMQ。

知识扩展：发布订阅和消息队列模式都支持系统解耦，两者是否一致呢?

概括地说，发布订阅和消息队列模式虽然都支持系统解耦，但它们在实现时采用的数据结构和方式并不相同。

首先，我们看一下它们实现解耦的数据结构。

> 发布订阅模式采用了消息中心，消息队列模式采用了消息队列中心，它们均用来存储生产者发布的数据，并均有主题、Broker 等概念；
>
> 唯一不同之处，是消息队列模式中采用了具有先进先出特征的队列结构进行存储，而订阅发布采用了 map 或数组等方式存储。

然后，我们再看看它们实现解耦的方式。

> 消息队列模式中，生产者发布数据到消息队列中心，消息队列中心会存储数据，等待消费者按需获取数据。这样生产者就不需要和消费者进行直接通信了，实现了生产者和消费者的解耦。
>
> 而在发布订阅模式中，消费者需要提前向消息中心订阅自己感兴趣的数据，待生产者发布数据到消息中心后，消息中心根据订阅者订阅信息将数据主动推送给消费者，也实现了消费者和生产者的解耦。

对于消息队列模式，消息队列中心无需提前获取消费者信息，因此对消费者比较灵活，适合消费者为临时用户的场景；而发布订阅模式，需要消费者提前向消息中心订阅消息，也就是说消息中心需要提前获取消费者信息，比较适合消费者为长驻进程或服务的场景。

总结

今天，我主要与你分享的是分布式通信技术中的消息队列模式。

首先，我通过用户注册的案例，与你介绍了什么是消息队列模式，以及它的好处。其中，消息队列模式中的核心是以一种具有先进先出特点的队列结构来存储数据，实现组件间的解耦和异步执行。

然后，我与你介绍了消息队列的基本原理，并以 RocketMQ 为例对其架构、核心组件和工作原理做了更深入的讲解，以帮助你进一步了解消息队列模型。

最后，我再通过一张思维导图来归纳一下今天的核心知识点吧。

加油，行动起来，试着将发布订阅和消息队列这两种通信模式用到你的业务场景中吧，相信你可以的。如果你需要进一步了解这两种通信模式对应的产品源码的话，相信你结合这两篇文章中讲述的原理，可以比较容易地开启你的源码之旅了。

思考题

消息队列模型中，消费者是主动去消息队列获取消息的，而消息队列需要保证多个消费者可以获取到消息，也就是说一个消费者获取消息后并不会删除该消息，那么如何保证同一个消息不被同一个消费者重复消费呢？

我是聂鹏程，感谢你的收听，欢迎你在评论区给我留言分享你的观点，也欢迎你把这篇文章分享给更多的朋友一起阅读。我们下期再会！

精选留言 (21)

忆水寒
2019-11-11

这个问题其实问的是幂等性问题，在目前流行的几种MQ中都可能存在。比如Kafka，在kafka实际上有个offset的概念，就是每个消息写进去，都有一个offset，代表他的序号，然后consumer消费了数据之后，每隔一段时间，会把自己消费过的消息的offset提交一下，代表我已经消费过了，下次我要是重启啥的，你就让我继续从上次消费到的offset来继续消费吧。但是凡事总有意外，比如我们之前生产经常遇到的，就是你有时候重启系统，看你怎么重启了，有时候直接kill进程再重启。这会导致consumer有些消息处理了，但是没来得及提交offset，尴尬了。重启之后，少数消息会再次消费一次。其实重复消费不可怕，可怕的是你没考虑到重复消费之后，怎么保证幂等性。

共 3 条评论 >　　32

信xin_n
2019-11-12

好像有点明白了，发布订阅模式是 push 模式，消息队列是 pull 模式。

共 1 条评论 >　　11

随心而至
2019-11-11

分布式数据存储，好像都是利用某种hash算法将数据存在不同的机器上。Kafka： hash（消息的键）来确定分区；Redis：hash（key）来确定slot；ES：hash（docId）来确定shard。

8

观弈道人
2019-11-22

感觉这个有点问题吧，不管是订阅还是队列模式，都是broker"送货上门"给消费端的，希望聂老师能再解释下，谢谢！

6

阿卡牛
2019-11-11

我理解的消息队列模型，同一份消息只能被消费一次吧？

共 2 条评论 >

3

piboye
2020-05-06

订阅模式的核心是一个消息可以被多个人消费，消息队列是一个消息一个人消费。

> 作者回复: 订阅模式除了一个消息可以被多个人消费，还有两点与消息队列不一样的地方，1）消费者需要提前订阅自己关注的消息；2）订阅发布中心根据订阅者订阅信息推送消息，在消息队列中是消费者主动获取消息。

共 2 条评论 >

2

青莲
2019-11-23

每个消费者都有维护一个自己消费消息的偏移量，可以存储在本地，或类似zookeepr的协同服务上，每次重启都去协同服务上拉取上一次消费的偏移量，从上一次记录开始消息；当然这种情况，由于各种异常原因还是有可能消息至少被消费一次，如常见的消息重放等，所以需要在业务方消费时进行业务上的幂等处理

2

小李飞菜刀
2020-11-20

感觉上一课的发布订阅模式,举了Kafka的例子不好，因为Kafka是拉的模式，是消费者主动消费数据，不是被推送过来的。
kafka的生产者只负责生产数据，不管有没有消费者；
kafka的消费者可以从头、上次消费点、最新的方式消费数据；

1

88591
2020-04-03

记录消费消息的对应关系。可以在客户端记录，也可以在服务端记录。，服务端记录更可靠，但是通信成本高。增加了服务器的开销。客户端记录简单，但是容易丢失。

作者回复: 根据不同场景，对数据可靠性的要求不一样，选择在客户端还是服务端

1

趁早
2021-09-04

规定一个queue or patition 在同一个消费组只能有一个消费者去消费

GaGi
2021-04-12

RocketMQ虽然是消息队列的中间件，但是其还是使用发布订阅模式实现，相对应的存储使用了队列。

penny19
2021-03-10

可不可以理解为，发布订阅的时延比队列模式低？因为队列一般去轮训，效率不如发布订阅的主动推过来

钱
2020-02-18

如何保证同一个消息不被同一个消费者重复消费呢?
有个小游戏，通过一个小窗口可以看到里面的动画内容，看的人需要买票，一张票只能看一次，看过之后票就作废了，如果有N个窗口，一张票可以从头看到尾，怎么保证每个窗口只看

一次呢？那就需要记住持票人看到那个位置的窗口了，我觉得MQ的解决思路是一样的，总之需要记录下每个消费者消费了什么消息，把每个消费者消费的消息都记录下来数据量太大了，由于是队列可以利用他的方向性，记录消费消息的位置，这样也能间接知道对应的消费者消费了那些消息。


小孩
2019-12-12

听下来感觉一个是push一个是pull，可是我用消息队列也是push啊，学完有点晕了

共 1 条评论 >

一毛钱
2019-11-30

kafka的offset就是解决多消费者的问题的


易儿易
2019-11-29

思考题:不是可以使用线程安全的队列吗？数据结构里是支持的，产品型的消息队列中间件不支持吗?


kylexy_0817
2019-11-28

在消费端作幂等处理，如在缓存上记录消息是否有被消费过，或在数据库的层面加上唯一索引约束数据只能被插入一次


梅子黄时雨
2019-11-11

可以在也许层面做防止重复消费的逻辑


tracy
2019-11-11

1.消息端实现幂等性，比如支付消息，以订单号为唯一标识，消息时候可以根据状态判断是否

处理过，数据库加上唯一索引
2.消费时候可以再内存中加set去重
3.应该需要根据不同的业务分析处理


Jackey
2019-11-11

这里想到可以考虑从consumer端控制，consumer消费消息时，broker可以返回一个偏移量，由consumer记录这个偏移量。下次消费时直接从偏移位置处开始消费。
或者由broker来记录消费者和对应偏移量，消费者要重复消费时直接不返回数据。但broker之间的数据同步应该会使系统更加复杂

共 1 条评论 >