图文 51 精益求精：深入研究一下Broker是如何持久化存储消息的?

930 人次阅读 2019-12-11 07:00:00

详情 评论

精益求精：深入研究一下Broker是如何持久化存储消息的?

石杉老哥重磅力作：《互联网java工程师面试突击》（第3季）**【强烈推荐】**：

全程真题驱动，精研Java面试中**6大专题的高频考点**，从面试官的角度剖析面试

（点击下方蓝字试听）

《互联网Java工程师面试突击》（第3季）

1、为什么Broker数据存储是最重要的一个环节?

小猛上次给大家分享完Producer的工作原理之后，团队整体都对RocketMQ的数据分片机制以及发送消息的时候如何写入各个Broker机器有了一定的了解。接着小猛就开始来给大家分享最为重要的Broker数据存储的环节。

首先我们得明确一点，为什么Broker数据存储是最重要的一个环节?

很简单，实际上类似RocketMQ、Kafka、RabbitMQ的消息中间件系统，他们不只是让你写入消息和获取消息那么简单，他们本身最重要的就是提供强大的数据存储能力，可以把亿万级的海量消息存储在自己的服务器的磁盘上。

这样的话，各种不同的系统从MQ中消费消息的时候，才可以从MQ服务器的磁盘中读取到自己需要的消息。

否则如果MQ不在机器磁盘上存储大量的消息，如果消息都放在自己的内存里，一个是内存很可能放不下，另外一个是可能你机器重启，内存里的消息就会全部丢失了。

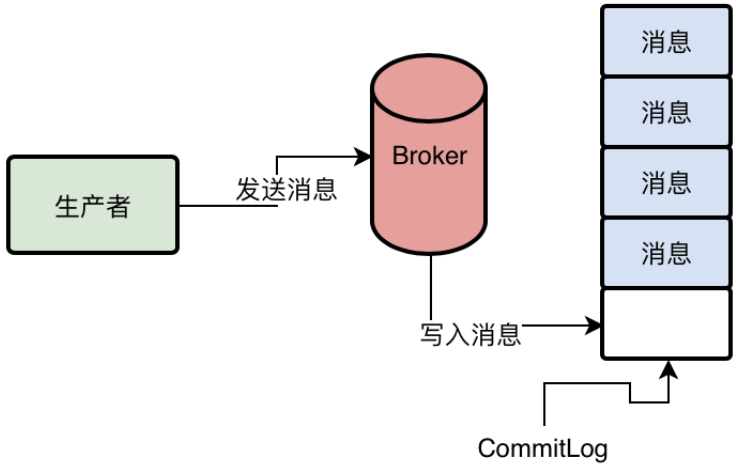
所以大家首先要明确一点，**Broker数据存储实际上才是一个MQ最核心的环节**，他决定了生产者消息写入的吞吐量，决定了消息不能丢失，决定了消费者获取消息的吞吐量，这些都是由他决定的。

所以今天我们来深入的探索一下Broker的数据存储机制。

2、CommitLog消息顺序写入机制

首先我们来思考一下，当生产者的消息发送到一个Broker上的时候，他接收到了一条消息，接着他会对这个消息做什么事情?

首先第一步，他会把这个消息直接写入磁盘上的一个日志文件，叫做CommitLog，直接顺序写入这个文件，如下图。



这个CommitLog是很多磁盘文件，每个文件限定最多1GB，Broker收到消息之后就直接追加写入这个文件的末尾，就跟上面的图里一样。如果一个CommitLog写满了1GB，就会创建一个新的CommitLog文件。

3、MessageQueue在数据存储中是体现在哪里呢?

接着我们会发现一个问题，如果写入这个Broker的消息都是进入到CommitLog中去存储的，那么上次我们提到的MessageQueue是体现在哪里的呢?

其实在Broker中，对Topic下的每个MessageQueue都会有一系列的ConsumeQueue文件。

这是什么意思呢?

就是在Broker的磁盘上，会有下面这种格式的一系列文件:

$HOME/store/consumequeue/{topic}/{queueId}/{fileName}

上面那一串东西是什么意思?

我们之前说过，对每个Topic你不是在这台Broker上都会有一些MessageQueue吗?所以你会看到，{topic}指代的就是某个Topic，{queueId}指代的就是某个MessageQueue。

然后对存储在这台Broker机器上的Topic下的一个MessageQueue，他有很多的ConsumeQueue文件，这个ConsumeQueue文件里存储的是一条消息对应在CommitLog文件中的offset偏移量。

很多人可能看到这里就直接看晕了，没明白这个是什么意思。。。

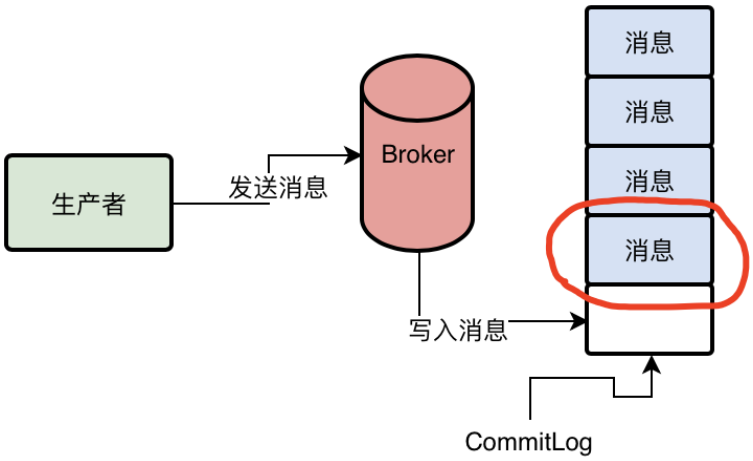
没关系，我们一步一图来给大家说明一下这是怎么回事。

首先我们假设有一个Topic，他有4个MessageQueue，然后在两台Broker机器上，每台Broker机器会存储两个MessageQueue。

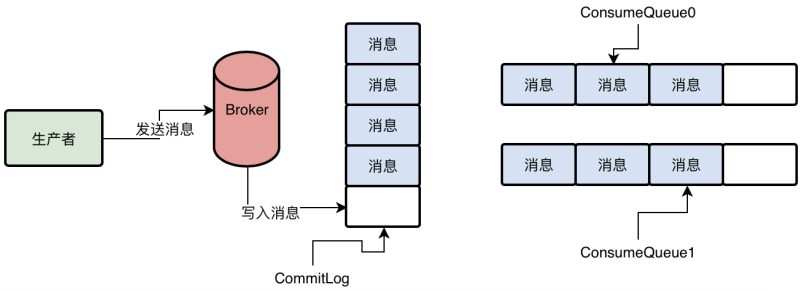
那么此时假设生产者选择对其中一个MessageQueue写入了一条消息，此时消息会发送到Broker上。

然后Broker必然会把这个消息写入自己的CommitLog文件中，是不是?

好，我们看下面的图里，我用红圈画出来了一个消息，我们假设就是刚刚写入的消息。



我们继续看下面的图，我在图里加入了两个ConsumeQueue，分别叫做ConsumeQueue0和ConsumeQueue1，他们分别对应着Topic里的MessageQueue0和MessageQueue1。



也就是说，Topic下的MessageQueue0和MessageQueue1就放在这个Broker机器上，而且他们每个MessageQueue目前在磁盘上就对应了一个ConsumeQueue，所以就是MessageQueue0对应着Broker磁盘上的ConsumeQueue0，MessageQueue1对应着磁盘上的ConsumeQueue1。

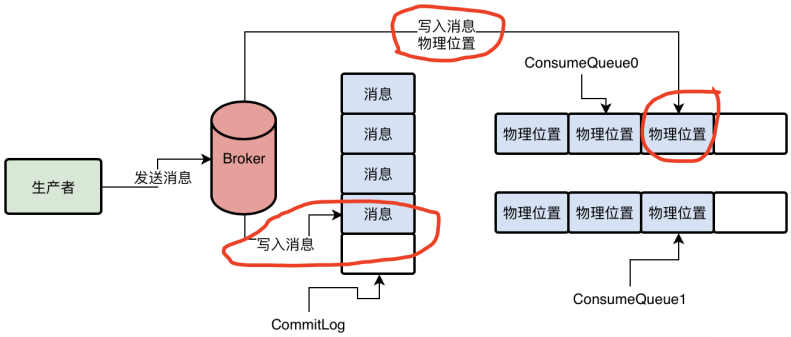
接着假设Queue的名字叫做：TopicOrderPaySuccess，那么此时在Broker磁盘上应该有如下两个路径的文件：

$HOME/store/consumequeue/TopicOrderPaySuccess/MessageQueue0/ConsumeQueue0磁盘文件

$HOME/store/consumequeue/TopicOrderPaySuccess/MessageQueue1/ConsumeQueue1磁盘文件

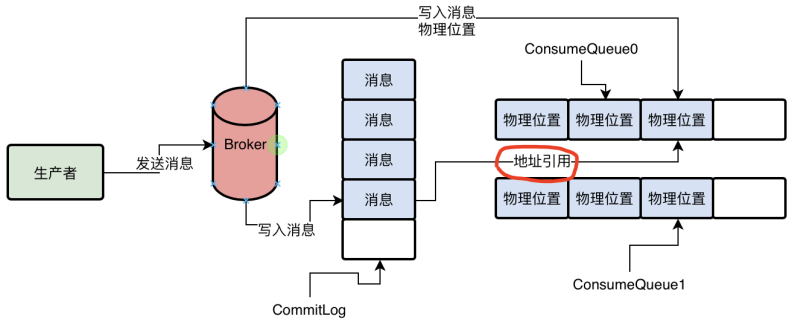
然后呢，当你的Broker收到一条消息写入了CommitLog之后，其实他同时会将这条消息在CommitLog中的物理位置，也就是一个文件偏移量，就是一个offset，写入到这条消息所属的MessageQueue对应的ConsumeQueue文件中去。

比如现在这条消息在生产者发送的时候是发送给MessageQueue0的，那么此时Broker就会将这条消息在CommitLog中的offset偏移量，写入到MessageQueue0对应的ConsumeQueue0中去，如下图所示。



所以实际上，ConsumeQueue0中存储的是一个一个消息在CommitLog文件中的物理位置，也就是offset

所以其实大家看下面的图，图里展示出来的是ConsumeQueue中的一个物理位置其实是对CommitLog文件中一个消息的引用。



实际上在ConsumeQueue中存储的每条数据不只是消息在CommitLog中的offset偏移量，还包含了消息的长度，以及tag hashcode，一条数据是20个字节，每个ConsumeQueue文件保存30万条数据，大概每个文件是5.72MB。

所以实际上Topic的每个MessageQueue都对应了Broker机器上的多个ConsumeQueue文件，保存了这个MessageQueue的所有消息在CommitLog文件中的物理位置，也就是offset偏移量。

4、如何让消息写入CommitLog文件近乎内存写性能的?

接着我们给大家讲一个比较关键的概念：对于生产者把消息写入到Broker时，Broker会直接把消息写入磁盘上的CommitLog文件，那么Broker是如何提升整个过程的性能的呢?

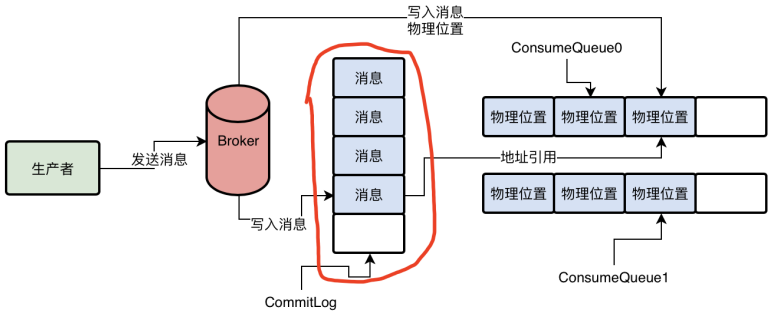
因为这个部分的性能提升会直接提升Broker处理消息写入的吞吐量，比如你写入一条消息到CommitLog磁盘文件假设需要10ms，那么每个线程每秒可以处理100个写入消息，假设有100个线程，每秒只能处理1万个写入消息请求。

但是如果你把消息写入CommitLog磁盘文件的性能优化为只需要1ms，那么每个线程每秒可以处理1000个消息写入，此时100个线程每秒可以处理10万个写入消息请求。所以大家可以明显看到，Broker把接收到的消息写入CommitLog磁盘文件的性能，对他的TPS有很大的影响。

所以在这里，Broker是基于OS操作系统的**PageCache**和**顺序写**两个机制，来提升写入CommitLog文件的性能的。

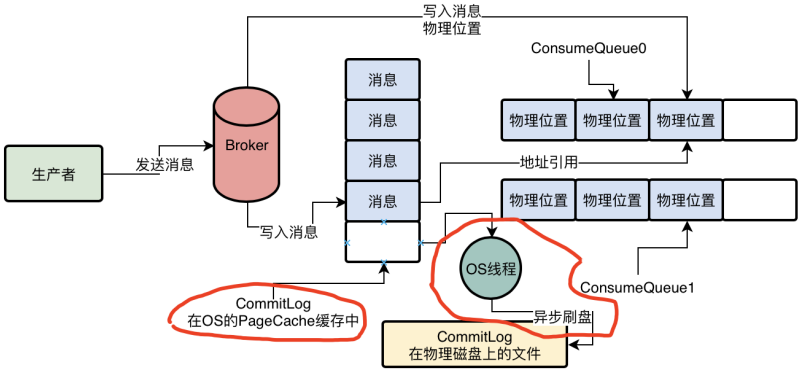
首先Broker是以顺序的方式将消息写入CommitLog磁盘文件的，也就是每次写入就是在文件末尾追加一条数据就可以了，对文件进行顺序写的性能要比对文件随机写的性能提升很多

我们看下面图里的红圈，就是示意数据是顺序写入的。



另外，数据写入CommitLog文件的时候，其实不是直接写入底层的物理磁盘文件的，而是先进入OS的PageCache内存缓存中，然后后续由OS的后台线程选一个时间，异步化的将OS PageCache内存缓冲中的数据刷入底层的磁盘文件。

我们看下面的图，图里示意出了，数据先写入OS的PageCache缓存中，然后后续由OS自己的线程将缓存里的数据刷入磁盘中。



所以在这样的优化之下，采用**磁盘文件顺序写+OS PageCache写入+OS异步刷盘的策略**，基本上可以让消息写入CommitLog的性能跟你直接写入内存里是差不多的，所以正是如此，才可以让Broker高吞吐的处理每秒大量的消息写入。

5、同步刷盘与异步刷盘

想必很多朋友此时可能意识到一个问题了，那么如果采用上述的模式，不就是异步刷盘的模式吗?

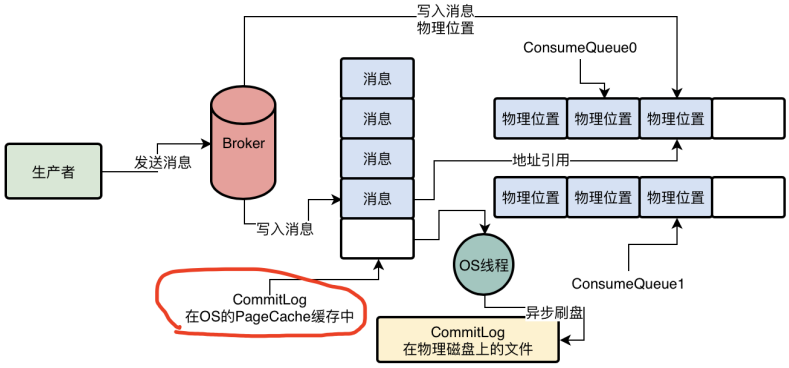
对的，在上述的异步刷盘模式下，生产者把消息发送给Broker，Broker将消息写入OS PageCache中，就直接返回ACK给生产者了。

此时生产者就认为消息写入成功了，那么会有什么问题吗?

问题肯定是有的，如果生产者认为消息写入成功了，但是实际上那条消息此时是在Broker机器上的os cache中的，如果此时Broker直接宕机，那么是不是os cache中的这条数据就会丢失了?

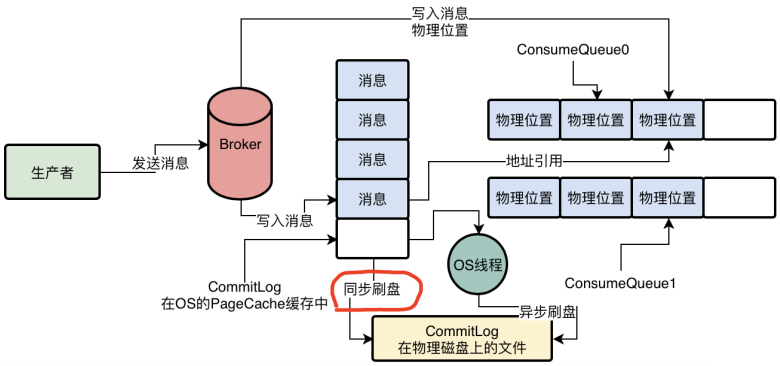
我们看下面的图，红圈圈出来了数据早os cache里的情况，如果此时broker宕机，那么必然导致这里的数据丢失，而producer还以为数据已经写入成功了，以为不会丢失，所以肯定是有问题的。

所以异步刷盘的的策略下，可以让消息写入吞吐量非常高，但是可能会有数据丢失的风险，这个是大家需要清除的。



另外一种模式叫做同步刷盘，如果你使用同步刷盘模式的话，那么生产者发送一条消息出去，broker收到了消息，必须直接强制把这个消息刷入底层的物理磁盘文件中，然后才会返回ack给producer，此时你才知道消息写入成功了。

只要消息进入了物理磁盘上，那么除非是你的物理磁盘坏了导致数据丢失，否则正常来说数据就不会丢失了，我们看下面的图，就是示意了同步刷盘的效果。



如果broker还没有来得及把数据同步刷入磁盘，然后他自己挂了，那么此时对producer来说会感知到消息发送失败了，然后你只要不停的重试发送就可以了，直到有slave broker切换成master broker重新让你可以写入消息，此时可以保证数据是不会丢的。

但是如果你强制每次消息写入都要直接进入磁盘中，必然导致每条消息写入性能急剧下降，导致消息写入吞吐量急剧下降，但是可以保证数据不会丢失。

好了，今天主要是分析一下broker对数据是如何存储的，从原理角度带着大家一步一图来分析一下，只有具体如何切换异步刷盘和同步刷盘的一些配置，后续我们会结合业务场景下的数据0丢失方案来讲解的。

6、对今天内容的一点小小总结

今天我们讲了broker最为核心的数据存储机制，包括如下一些知识点：

为什么Broker数据存储机制是一个MQ最为核心的环节？
CommitLog数据存储机制
MessageQueue对应的ConsumeQueue物理位置存储机制
基于CommitLog顺序写+OS Cache+异步刷盘的高吞吐消息写入的机制
同步刷盘和异步刷盘各自的优缺点：高吞吐写入+丢失数据风险，写入吞吐量下降+数据不丢失

End

狸猫技术窝其他精品专栏推荐：

《从零开始带你成为JVM实战高手》
《21天Java 面试突击训练营》（分布式篇）（现更名为：**互联网Java工程师面试突击第2季**）
互联网Java工程师面试突击（第1季）

重要说明：

如何提问：每篇文章都有评论区，大家可以尽情在评论区留言提问，我会逐一答疑

如何加群：购买了狸猫技术窝专栏的小伙伴都可以加入**狸猫技术交流群**

具体加群方式，请参见目录菜单下的文档：《付费用户如何加群？》**（购买后可见）**