原 探秘RocketMQ消息持久化

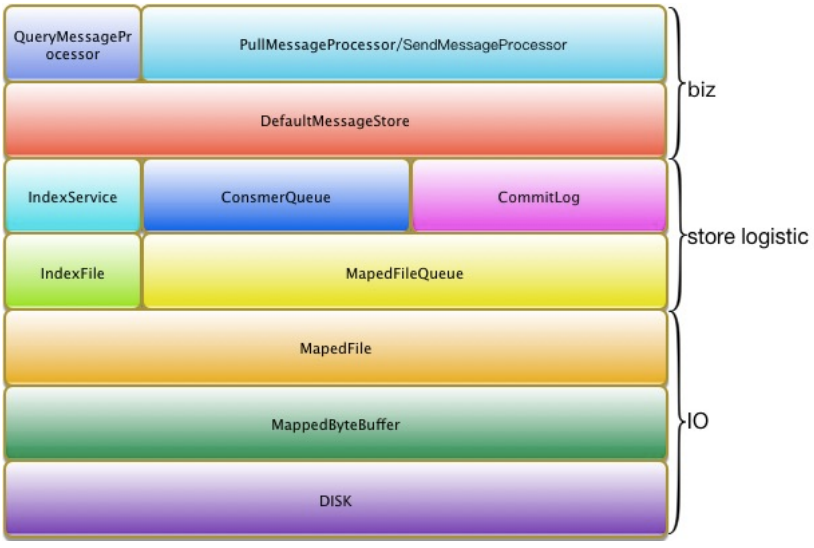
摘要: 之前对RocketMq的上层接口进行过介绍，但是作为一个可持久化的MQ中间件，那么其核心必然是对消息的持久化这一块。这也是我一直想去了解一下，最近稍微闲下来一点，把RocketMQ这一块代码实现给过了一下，这里对看到得内容和想法进行一次总结。这里分三部分来介绍RocketMQ消息持久化，分别是来自消息提供端的写，来自消费端的读以及从本地磁盘中进行恢复。

探秘RocketMQ消息持久化

之前对RocketMq的上层接口进行过介绍，但是作为一个可持久化的MQ中间件，那么其核心必然是对消息的持久化这一块。这也是我一直想去了解一下，最近稍微闲下来一点，把RocketMQ这一块代码实现给过了一下，这里对看到得内容和想法进行一次总结。这里分三部分来介绍RocketMQ消息持久化，分别是来自消息提供端的写，来自消费端的读以及从本地磁盘中进行恢复。

在介绍之前先了解一下RocketMQ中对数IO的统一入口，分别是`MapedFileQueue`和`MapedFile`。`MapedFileQueue`是对某个目录下面文件IO的统一入口，而`MapedFile`是对`MapedFileQueue`目录下某个文件的IO封装（具体是通过`MapedByteBuffer`来进行IO操作），对于`MapedFileQueue`和`MapedFile`来说，它们关注的只是往哪个位置些多少数据，以及从哪个位置读取数据，而不管里面存储的内容，写入的内容都将是`byte`数组。RocketMQ的消息文件以及其他文件的IO都是基于这两个类来做，而不是直接操作IO。介绍完这个之后，那么下面将对RocketMQ如何基于`MapedFileQueue`和`MapedFile`来做到消息的写和读。

在介绍之前，先看看RocketMQ的broker的整体架构图



来自消息提供端的写

消息提供端发起一个`send`操作，会被`broker`中`SendMessageProcessor`所处理，至于`SendMessageProcessor`这个类中做了哪些事情，这里就不做解释，主要是对`SendMessageProcessor`如何将消息写入到磁盘进行介绍，`SendMessageProcessor`会把写磁盘的操作交给`DefaultMessageStore`类去处理，而`DefaultMessageStore`也不会做具体IO的事情，而是交给`CommitLog`，在`CommitLog`之下则是`MapedFileQueue`，在`MapedFileQueue`中会写入到最新的`MapedFile`中（此时的`MapedFile`默认最大1G，所有存储的配置都在`MessageStoreConfig`类中获取。），这里从`SendMessageProcessor`到`DefaultMessageStore`再到`CommitLog`，最后到`MapedFileQueue`，这个过程中是所有的topic都操作同一个`MapedFileQueue`,那就是说所有的Topic的

消息都些在一个目录下面（因为一个MapedFileQueue对应一个目录，CommitLog的目录默认是在${user_home}/store/commitlog下），上面由消息提供端每次send都是一个完整的消息体，那就是一个完整的消息，这个消息体将会连续的写到MapedFileQueue的最新MapedFile中，在MapedFileQueue里面维护了commitlog的全局offset，那么只需要告诉MapedFileQueue一个全局offset和消息体的大小，那么就可以从MapedFileQueue中读取一个消息。但是在commitlog中只是负责将消息写入磁盘，而不管你怎么来读取，但是CommitLog通过MapedFileQueue写完之后，那么会得到当前写的位置，以及消息体大小，同时加上topic的元数据信息，通过异步队列的方式写到topic的索引文件，这个文件就是下面介绍消息读取的时候用到。

在CommitLog的putMessage方法调用MapedFileQueue写完消息之后，那么会调用DefaultMessageStore的putDispatchRequest方法进行将本次写操作广播出去，具体代码如下：

```
public PutMessageResult putMessage(final MessageExtBrokerInner msg) {
            ..........
            result = mapedFile.appendMessage(msg, this.appendMessageCallback);
            ..........
            DispatchRequest dispatchRequest = new DispatchRequest(//
                topic,// 1
                queueId,// 2
                result.getWroteOffset(),// 3
                result.getWroteBytes(),// 4
                tagsCode,// 5
                msg.getStoreTimestamp(),// 6
                result.getLogicsOffset(),// 7
                msg.getKeys(),// 8
                /**
                 * Transaction
                 */
                msg.getSysFlag(),// 9
                msg.getPreparedTransactionOffset());// 10

            this.defaultMessageStore.putDispatchRequest(dispatchRequest);
            ..........
        return putMessageResult;
    }
```

如果继续跟进putDispatchRequest代码，就会发现是将dispatchRequest放到一个队列，然后由另一个线程去处理这个数据，这样可以提高消息提供端写入broker的效率，在这个线程中，会触发DefaultMessageStore的putMessagePostionInfo方法，该方法实现如下：

```
public void putMessagePostionInfo(String topic, int queueId, long offset, int size, long tagsCode,
            long storeTimestamp, long logicOffset) {
        ConsumeQueue cq = this.findConsumeQueue(topic, queueId);
        cq.putMessagePostionInfoWrapper(offset, size, tagsCode, storeTimestamp, logicOffset);
    }
```

可以看到通过topic和queueId得到具体的ConsumerQueue,确定这个消息在哪个消费队列里面，同事触发cq.putMessagePostionInfoWrapper调用，从方法名就知道是记录消息位置的，最后会调用ConsumerQueue的putMessagePostionInfo方法，这个方法实现如下：

```
private boolean putMessagePostionInfo(final long offset, final int size, final long tagsCode,
            final long cqOffset) {
        // 在数据恢复时会走到这个流程
        if (offset <= this.maxPhysicOffset) {
            return true;
        }

        this.byteBufferIndex.flip();
        this.byteBufferIndex.limit(CQStoreUnitSize);
        this.byteBufferIndex.putLong(offset);
        this.byteBufferIndex.putInt(size);
        this.byteBufferIndex.putLong(tagsCode);

        final long expectLogicOffset = cqOffset * CQStoreUnitSize;

        MapedFile mapedFile = this.mapedFileQueue.getLastMapedFile(expectLogicOffset);
        if (mapedFile != null) {
            // 纠正MapedFile逻辑队列索引顺序
            if (mapedFile.isFirstCreateInQueue() && cqOffset != 0 && mapedFile.getWrotePostion() == 0) {
                this.minLogicOffset = expectLogicOffset;
                this.fillPreBlank(mapedFile, expectLogicOffset);
                log.info("fill pre blank space " + mapedFile.getFileName() + " " + expectLogicOffset + " "
                        + mapedFile.getWrotePostion());
            }

            if (cqOffset != 0) {
```

```
                long currentLogicOffset = mapedFile.getWrotePostion() + mapedFile.getFileFromOffset();
                if (expectLogicOffset != currentLogicOffset) {
                    // XXX: warn and notify me
                    logError
                        .warn(
                            "[BUG]logic queue order maybe wrong, expectLogicOffset: {} currentLogicOffset: {}
                            expectLogicOffset, //
                            currentLogicOffset,//
                            this.topic,//
                            this.queueId,//
                            expectLogicOffset - currentLogicOffset//
                        );
                }
            }

            // 记录物理队列最大offset
            this.maxPhysicOffset = offset;
            return mapedFile.appendMessage(this.byteBufferIndex.array());
        }

        return false;
    }
```

下面是要写入磁盘的内容

```
this.byteBufferIndex.flip();
this.byteBufferIndex.limit(CQStoreUnitSize);
this.byteBufferIndex.putLong(offset);
this.byteBufferIndex.putInt(size);
this.byteBufferIndex.putLong(tagsCode);
```

可以看到是写入了commitlog得全局offset和消息体的大小，以及tags信息。

```
MapedFile mapedFile = this.mapedFileQueue.getLastMapedFile(expectLogicOffset);
```

上面看到，ConsumerQueue(默认情况下ConsumerQueue是在${user_home}/store/consumerqueue/${queueId})也是通过MapedFileQueue来进行磁盘IO的，素以MapedFileQueue可以理解为RocketMQ的磁盘访问入口。

到这里基本上完成了一次消息提供端发起send操作所做的哪些事情，主要是通过CommitLog来进行消息内容的持久化，以及通过ConsumerQueue来确定消息被哪个队列消费，以及消息的索引持久化。

这里再介绍一下上面的queueId是怎么来的，因为消息提供端send某个topic的消息并不知道queueId,这个queueId是在broker端生成的，生成代码在SendMessageProcessor的方法consumerSendMsgBack中，代码段如下：

```
int queueIdInt =
                Math.abs(this.random.nextInt() % 99999999) % subscriptionGroupConfig.getRetryQueueNums();
```

来自消息消费端的读

在broker端处理来自消费端的读请求，是交给PullMessageProcessor类来处理，在方法processRequest经过一系列处理之后，会交给DefaultMessageStore的getMessage方法，我这里贴出该方法主要代码段

```
    public GetMessageResult getMessage(final String group, final String topic, final int queueId,
            final long offset, final int maxMsgNums, final SubscriptionData subscriptionData) {
        ........
        GetMessageResult getResult = new GetMessageResult();

        // 有个读写锁，所以只访问一次，避免锁开销影响性能
        final long maxOffsetPy = this.commitLog.getMaxOffset();

        ConsumeQueue consumeQueue = findConsumeQueue(topic, queueId);
        if (consumeQueue != null) {
            minOffset = consumeQueue.getMinOffsetInQuque();
            maxOffset = consumeQueue.getMaxOffsetInQuque();
            .........//逻辑校验
            else {
                //这里的offset是只从第几个消息开始消费，该方法返回的时从offset之后的消息体索引的io
                SelectMapedBufferResult bufferConsumeQueue = consumeQueue.getIndexBuffer(offset);
                if (bufferConsumeQueue != null) {
                    try {
                        status = GetMessageStatus.NO_MATCHED_MESSAGE;

                        long nextPhyFileStartOffset = Long.MIN_VALUE;
                        long maxPhyOffsetPulling = 0;
```

```
                    int i = 0;
                    final int MaxFilterMessageCount = 16000;
                    boolean diskFallRecorded = false;
                    //循环读出多个消息内容
                    for (; i < bufferConsumeQueue.getSize() && i < MaxFilterMessageCount; i +=
                            ConsumeQueue.CQStoreUnitSize) {
                        //得到了一个消息体索引信息
                        long offsetPy = bufferConsumeQueue.getByteBuffer().getLong();//commitlog的全局偏移量
                        int sizePy = bufferConsumeQueue.getByteBuffer().getInt();//消息大小
                        long tagsCode = bufferConsumeQueue.getByteBuffer().getLong();//tag信息

                        maxPhyOffsetPulling = offsetPy;

                        //参数校验

                        // 消息过滤
                        if (this.messageFilter.isMessageMatched(subscriptionData, tagsCode)) {
                            //从commitlog中读取消息
                            SelectMapedBufferResult selectResult =
                                    this.commitLog.getMessage(offsetPy, sizePy);
                            .........
                        }
        ......

        getResult.setStatus(status);
        getResult.setNextBeginOffset(nextBeginOffset);
        getResult.setMaxOffset(maxOffset);
        getResult.setMinOffset(minOffset);
        return getResult;
    }
```

上面是读取消息的部分逻辑，可以看到是先从`ConsumerQueue`中获取消息索引，然后再从commitlog中读取消息内容。这些内容也是在存储消息的时候写入的。因为broker端并不是一直运行的，而里面的commitlog的offset是有状态的，不能说你的broker挂掉了，导致commitlog的offset丢失，可能导致消息被覆盖。所以下面再用上一小段来介绍RocketMQ如何做到commitlog的offset重启后不丢失。

从本地磁盘恢复

在`MapedFileQueue`中有一个`load`方法，这个方法是将`MapedFileQueue`所管理目录中得文件加载到`MapedFile`中，如果你追踪这个`load`方法的被调用链路，会发现是在`BrokerController`的`initialize`触发了整个调用，那就是说在broker启动的时候，会触发`CommitLog`去将本地磁盘的数据关系加载到系统里面来，上面说了`CommitLog`有一个全局offset，这个offset在broker启动的时候怎么被查找的呢？如果你们熟悉的话，应该猜得到，是将`CommitLog`的`MapedFileQueue`中文件进行计算，得到当前`CommitLog`的全局offset，下面我贴出具体找得代码：

```
/**
     * recover时调用，不需要加锁
     */
    public void truncateDirtyFiles(long offset) {
        List<MapedFile> willRemoveFiles = new ArrayList<MapedFile>();

        for (MapedFile file : this.mapedFiles) {
            long fileTailOffset = file.getFileFromOffset() + this.mapedFileSize;
            if (fileTailOffset > offset) {
                if (offset >= file.getFileFromOffset()) {
                    file.setWrotePostion((int) (offset % this.mapedFileSize));
                    file.setCommittedPosition((int) (offset % this.mapedFileSize));
                }
                else {
                    // 将文件删除掉
                    file.destroy(1000);
                    willRemoveFiles.add(file);
                }
            }
        }

        this.deleteExpiredFile(willRemoveFiles);
    }
```

上面代码是在`MapedFileQueue`中得，是被`CommitLog`的`recoverAbnormally`方法调用，而`recoverAbnormally`最上层触发也是在`BrokerController`的`initialize`方法中。上面说的是`CommitLog`的恢复过程，而`ConsumerQueue`的恢复恢复过程也是类似，感兴趣可以自己去看看。

分类：Java's life 字数：2259

标签： rocketmq 消息持久化

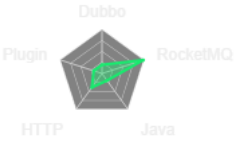
打赏 点赞 收藏 分享

+ 关注

Bieber

高级程序员 杭州

粉丝 178 | 博文 36 | 码字总数 83312 | 作品 1



相关博客

RocketMQ的模块
愤怒的lemon
136 0

RocketMQ安装、启动
站在巨人的肩膀上奋斗
269 2

RocketMQ-debug笔记
强子哥哥
146 0

评论 (0)

Ctrl+Enter 发表评论