考拉哥的博客
代码如诗，人生如歌。

RocketMQ学习（九）：顺序消息

发表于2015 年 12 月 6 日由考拉哥

rocketmq的顺序消息需要满足2点：

1.Producer端保证发送消息有序，且发送到同一个队列。
2.consumer端保证消费同一个队列。

先看个例子，代码版本跟前面的一样。
Producer类：

```
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;

import com.alibaba.rocketmq.client.exception.MQBrokerException;
import com.alibaba.rocketmq.client.exception.MQClientException;
import com.alibaba.rocketmq.client.producer.DefaultMQProducer;
import com.alibaba.rocketmq.client.producer.MessageQueueSelector;
import com.alibaba.rocketmq.client.producer.SendResult;
import com.alibaba.rocketmq.common.message.Message;
import com.alibaba.rocketmq.common.message.MessageQueue;
import com.alibaba.rocketmq.remoting.exception.RemotingException;


/**
 * Producer，发送顺序消息
 */
public class Producer {
    public static void main(String[] args) throws IOException {
        try {
            DefaultMQProducer producer = new DefaultMQProducer("please_rename_unique_

            producer.setNamesrvAddr("192.168.0.104:9876");

            producer.start();

            String[] tags = new String[] { "TagA", "TagC", "TagD" };

            Date date = new Date();
            SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
            String dateStr = sdf.format(date);
            for (int i = 0; i < 10; i++) {
                // 加个时间后缀
                String body = dateStr + " Hello RocketMQ " + i;
                Message msg = new Message("TopicTestjjj", tags[i % tags.length], "KEY

                SendResult sendResult = producer.send(msg, new MessageQueueSelector()
                    @Override
                    public MessageQueue select(List<MessageQueue> mqs, Message msg, O
                        Integer id = (Integer) arg;
                        return mqs.get(id);
                    }
                }, 0);//0是队列的下标

                System.out.println(sendResult + ", body:" + body);
            }

            producer.shutdown();
```

```
        } catch (MQClientException e) {
            e.printStackTrace();
        } catch (RemotingException e) {
            e.printStackTrace();
        } catch (MQBrokerException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.in.read();
    }
}
```

Consumer端：

```
import java.util.List;
import java.util.Random;
import java.util.concurrent.TimeUnit;

import com.alibaba.rocketmq.client.consumer.DefaultMQPushConsumer;
import com.alibaba.rocketmq.client.consumer.listener.ConsumeOrderlyContext;
import com.alibaba.rocketmq.client.consumer.listener.ConsumeOrderlyStatus;
import com.alibaba.rocketmq.client.consumer.listener.MessageListenerOrderly;
import com.alibaba.rocketmq.client.exception.MQClientException;
import com.alibaba.rocketmq.common.consumer.ConsumeFromWhere;
import com.alibaba.rocketmq.common.message.MessageExt;


/**
 * 顺序消息消费，带事务方式（应用可控制Offset什么时候提交）
 */
public class Consumer {

    public static void main(String[] args) throws MQClientException {
        DefaultMQPushConsumer consumer = new DefaultMQPushConsumer("please_rename_uni
        consumer.setNamesrvAddr("192.168.0.104:9876");
        /**
         * 设置Consumer第一次启动是从队列头部开始消费还是队列尾部开始消费<br>
         * 如果非第一次启动，那么按照上次消费的位置继续消费
         */
        consumer.setConsumeFromWhere(ConsumeFromWhere.CONSUME_FROM_FIRST_OFFSET);

        consumer.subscribe("TopicTestjjj", "TagA || TagC || TagD");

        consumer.registerMessageListener(new MessageListenerOrderly() {

            Random random = new Random();

            @Override
            public ConsumeOrderlyStatus consumeMessage(List<MessageExt> msgs, Consume
                context.setAutoCommit(true);
                System.out.print(Thread.currentThread().getName() + " Receive New Mes
                for (MessageExt msg: msgs) {
                    System.out.println(msg + ", content:" + new String(msg.getBody())
                }
                try {
                    //模拟业务逻辑处理中...
                    TimeUnit.SECONDS.sleep(random.nextInt(10));
                } catch (Exception e) {
                    e.printStackTrace();
                }
                return ConsumeOrderlyStatus.SUCCESS;
            }
        });

        consumer.start();

        System.out.println("Consumer Started.");
    }
```

```
55
56  }
```

NameServer和BrokerServer起来后，运行打印，把前面的不重要的去掉了，只看后面的几列：
content:2015-12-06 17:03:21 Hello RocketMQ 0
content:2015-12-06 17:03:21 Hello RocketMQ 1
content:2015-12-06 17:03:21 Hello RocketMQ 2
content:2015-12-06 17:03:21 Hello RocketMQ 3
content:2015-12-06 17:03:21 Hello RocketMQ 4
content:2015-12-06 17:03:21 Hello RocketMQ 5
content:2015-12-06 17:03:21 Hello RocketMQ 6
content:2015-12-06 17:03:21 Hello RocketMQ 7
content:2015-12-06 17:03:21 Hello RocketMQ 8
content:2015-12-06 17:03:21 Hello RocketMQ 9

可以看到，消息有序的。

如何在集群消费时保证消费的有序呢？

1.ConsumeMessageOrderlyService类的start()方法，如果是集群消费，则启动定时任务，定时向broker发送批量锁住当前正在消费的队列集合的消息，具体是consumer端拿到正在消费的队列集合，发送锁住队列的消息至broker，broker端返回锁住成功的队列集合。
consumer收到后，设置是否锁住标志位。
这里注意2个变量：
consumer端的RebalanceImpl里的ConcurrentHashMap processQueueTable，是否锁住设置在ProcessQueue里。
broker端的RebalanceLockManager里的ConcurrentHashMap> mqLockTable，这里维护着全局队列锁。

2.ConsumeMessageOrderlyService.ConsumeRequest的run方法是消费消息，这里还有个MessageQueueLock messageQueueLock，维护当前consumer端的本地队列锁。保证当前只有一个线程能够进行消费。

3.拉到消息存入ProcessQueue，然后判断，本地是否获得锁，全局队列是否被锁住，然后从ProcessQueue里取出消息，用MessageListenerOrderly进行消费。
拉到消息后调用ProcessQueue.putMessage(final List msgs) 存入，具体是存入TreeMap msgTreeMap。
然后是调用ProcessQueue.takeMessags(final int batchSize)消费，具体是把msgTreeMap里消费过的消息，转移到TreeMap msgTreeMapTemp。

4.本地消费的事务控制，ConsumeOrderlyStatus.SUCCESS（提交），ConsumeOrderlyStatus.SUSPEND_CURRENT_QUEUE_A_MOMENT（挂起一会再消费），在此之前还有一个变量ConsumeOrderlyContext context的setAutoCommit()是否自动提交。
当SUSPEND_CURRENT_QUEUE_A_MOMENT时，autoCommit设置为true或者false没有区别，本质跟消费相反，把消息从msgTreeMapTemp转移回msgTreeMap，等待下次消费。

当SUCCESS时，autoCommit设置为true时比设置为false多做了2个动作，
consumeRequest.getProcessQueue().commit()和
this.defaultMQPushConsumerImpl.getOffsetStore().updateOffset(consumeRequest.getMessageQueue(), commitOffset, false);
ProcessQueue.commit()：本质是删除msgTreeMapTemp里的消息，msgTreeMapTemp里的消息在上面消费时从msgTreeMap转移过来的。
this.defaultMQPushConsumerImpl.getOffsetStore().updateOffset()：本质是把拉消息的偏移量更新到本地，然后定时更新到broker。

那么少了这2个动作会怎么样呢，随着消息的消费进行，msgTreeMapTemp里的消息堆积越来越多，消费消息的偏移量一直没有更新到broker导致consumer每次重新启动后都要从头开始重复消费。
就算更新了offset到broker，那么msgTreeMapTemp里的消息堆积呢？不知道这算不算bug。
所以，还是把autoCommit设置为true吧。

此条目发表在编程语言分类目录，贴了java, rocketmq标签。将固定链接加入收藏夹。
