为什么会需要消息队列(MQ)?

主要原因是由于在高并发环境下,由于来不及同步处理,请求往往会发生堵塞,比如说,大量的insert,update之类的请求同时到达MySQL,直接导致无数的行锁表锁,甚至最后请求会堆积过多,从而触发too many connections错误。通过使用消息队列,我们可以异步处理请求,从而缓解系统的压力。

RocketMQ入门-几种使用方式

这里主要讲有序消费、广播消费、延时消费、批量消费

有序消费

RocketMQ的每个topic下会有多个queue,默认是4个,对于每个queue可以保证先进先出。

生产者角度

如果生产者直接发送消息到topic,消息可能会进入到任何一个队列导致无序,所以,对于需要顺序的消息,如一个用户连续3次付款操作,需要发到同一个队列中。

消费者角度

消费者在消费队列的时候需要按序消费,使用MessageListenerOrderly即可。 使用MessageListenerOrderly的消费者会定期去锁住topic的所有队列,保证只有它在消费,且在本地使用单线程操作来保证本地按序消费。

```
1 //生产者
2 //模拟100个用户
3 for(int userId=0;userId<100;userId++){
4    //每个用户连续3次付款
5    for(int payNum=0;payNum<3;payNum++){
6    Message msg = new Message();
7    msg.setTopic("payTopic");
8    msg.setBody(("用户" + userId + "第" + payNum + "次付款操作").getBytes());
9    producer.send(msg, new MessageQueueSelector() {
10    @Override</pre>
```

```
public MessageQueue select(List<MessageQueue> mqs, Message msg, Object
arg) {
   int index = (Integer)(arg) % mqs.size();
   return mqs.get(index);
   }
14
15
   }, userId);
   }
16
17 }
18 //消费者
19 consumer.registerMessageListener(new MessageListenerOrderly() {
    @Override
20
21 public ConsumeOrderlyStatus consumeMessage(List<MessageExt> msgs, Consu
meOrderlyContext context) {
  for(int i=0; i<msgs.size(); i++){</pre>
23 MessageExt msg = msgs.get(i);
24 System.out.println(msg.getTopic() + " " + msg.getTags() + " " + new Str
ing(msg.getBody()));
25
26
  return ConsumeOrderlyStatus.SUCCESS;
27
28 });
29 consumer.start();
```

广播消费

RocketMQ支持集群消费和广播消费。

集群消费

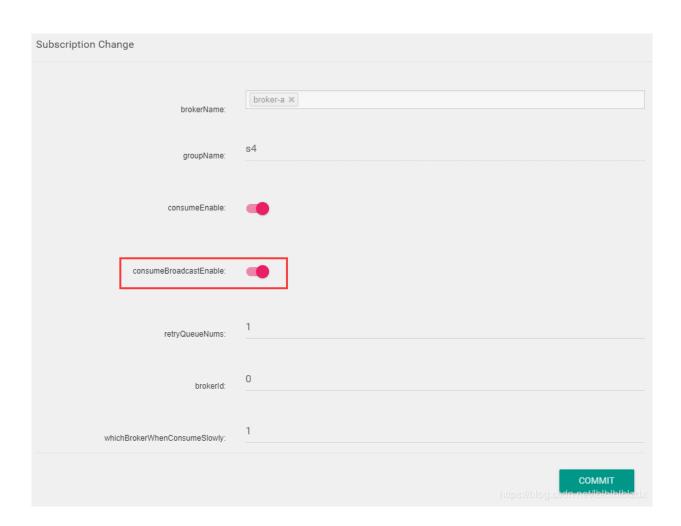
默认是集群消费模式,对于同一个消费组里的消费者,会分摊消息,如A消费了B就不能消费。

测试的时候发现,一个进程里无法创建一个消费组的两个消费者,会报错,必须分两个进程去创建,创建的时候groupName要一致,才会认为在同一个消费组。

广播消费

对于同一个消费组里的消费者,每个消费者都能收到每一份消息,AB能消费同一个消息,相当于广播。

测试的时候不小心在控制台将consumer的consumeBroadcastEnable设为false,导致无法消费集群消息,如果出现类似问题可以注意一下。



```
1 //groupName必须一致才认为在一个消费组里,同一个消费组里的消费者应该订阅一样的to
pic
2 DefaultMQPushConsumer consumer = new
DefaultMQPushConsumer("myGroupName");
3 //默认为集群模式,此处修改为广播模式
4 consumer.setMessageModel(MessageModel.BROADCASTING);
```

延时消费

RocketMQ不支持精准的延时,只能在broker.conf中设置时延级别 messageDelayLevel = 1s 5s 10s

在发送消息的时候设置时延级别

```
1 //根据配置,1代表1s,2代表5s,3代表10s,如果设置为0代表没有时延
2 //如下发送则会在5s后收到延时消息
3 msg.setDelayTimeLevel(2);
4 producer.send(msg);
```

批量消费

RocketMQ支持批量消费,但是批量的消息的topic需要一样,tag则不限制。

```
1 List<Message> list = new ArrayList<>();
2 list.add(new Message("mytopic", "tag1", "body".getBytes()));
3 list.add(new Message("mytopic", "tag2", "body".getBytes()));
4 producer.send(list);
```