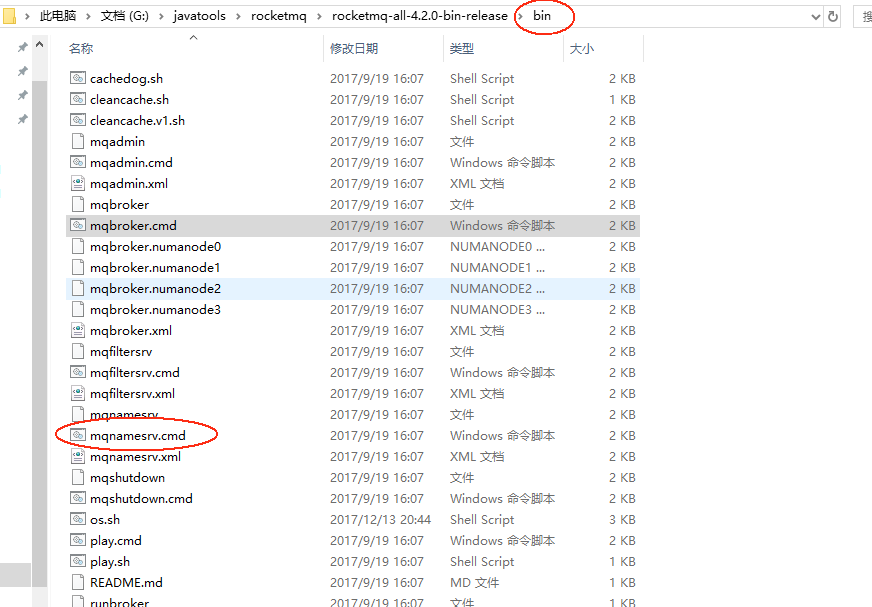
# <https://help.aliyun.com/knowledge_detail/29547.html>

# Windows下启动



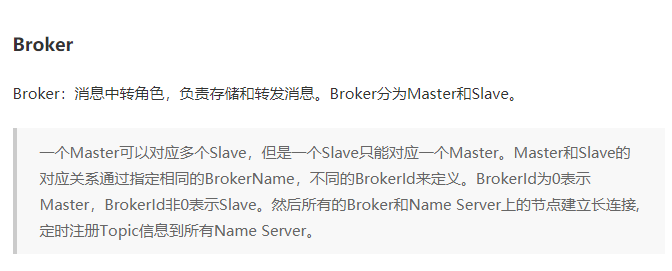
双击这个启动即可.

然后在该bin目录下,进入到cmd.输入如下命令启动mqbroker:

start mqbroker.cmd -n 127.0.0.1:9876 autoCreateTopicEnable=true

# 基本概念

595c617baa914b8090e07df117feaae1



# Topic和Tag的最佳实践



# 发送消息的三种方式

## 可靠同步发送

消息发送后，会收到接收方发回响应之后才会发送下一个消息。

SendResult sendResult = producer.send(msg);

## 可靠异步发送

发送数据后，不需要等待接收方发响应，接着发送下一个数据。异步发送，需要用户实现异步发送回调接口。发送方通过回调接口接收服务器响应，并对结果进行处理。

应用场景是：一般链路耗时较长，对RT响应时间较为敏感的业务场景。例如用户视频上传后通知启动转码服务。

producer.sendAsync(msg, new SendCallback() {

@Override

public void onSuccess(final SendResult sendResult) {

// 消费发送成功

System.out.println("send message success. topic=" + sendResult.getTopic() + ", msgId=" + sendResult.getMessageId());

}

@Override

public void onException(OnExceptionContext context) {

// 消息发送失败，需要进行重试处理，可重新发送这条消息或持久化这条数据进行补偿处理

System.out.println("send message failed. topic=" + context.getTopic() + ", msgId=" + context.getMessageId());

}

});

## 单向发送

发送方只负责发送消息，不等待服务器响应且没有回调函数触发，只发送请求不等待应答，此方式发送消息耗时非常短，一般为微妙级别。

应用场景：耗时非常短，但对可靠性要求不是特别高的场景，例如日志收集。

producer.sendOneway(msg);



# 多线程收发消息

生产者和消费者客户端对象都是线程安全的，可以在多个线程之间共享使用。

但是避免为每个线程创建一个客户端实例。

# 消息类型

消息类型的范围在topic下，一个topic对应一种消息类型。

## 普通消息

## 定时和延时消息

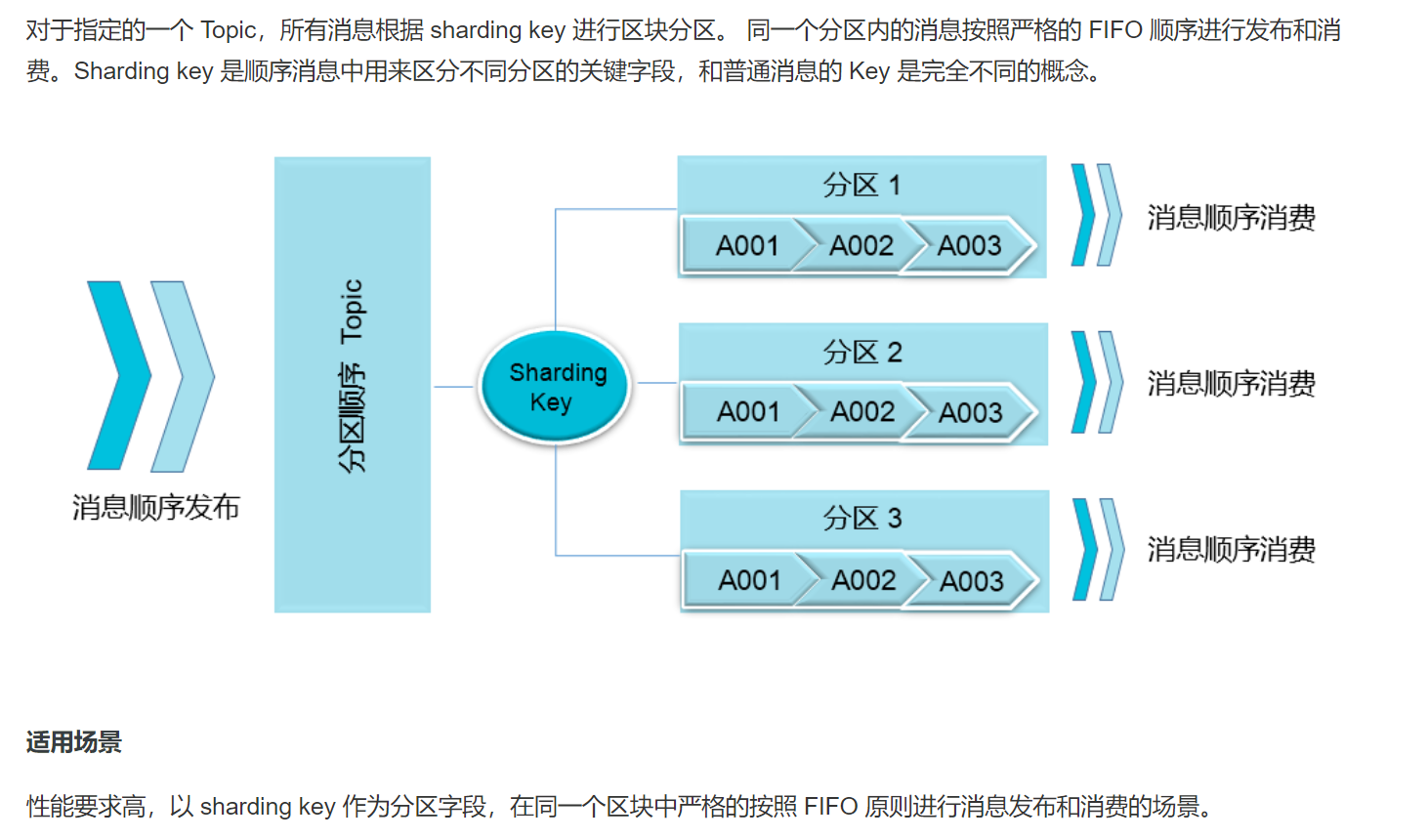
## 顺序消息

顺序消息又分为全局顺序消息和分区消息

全局顺序消息：



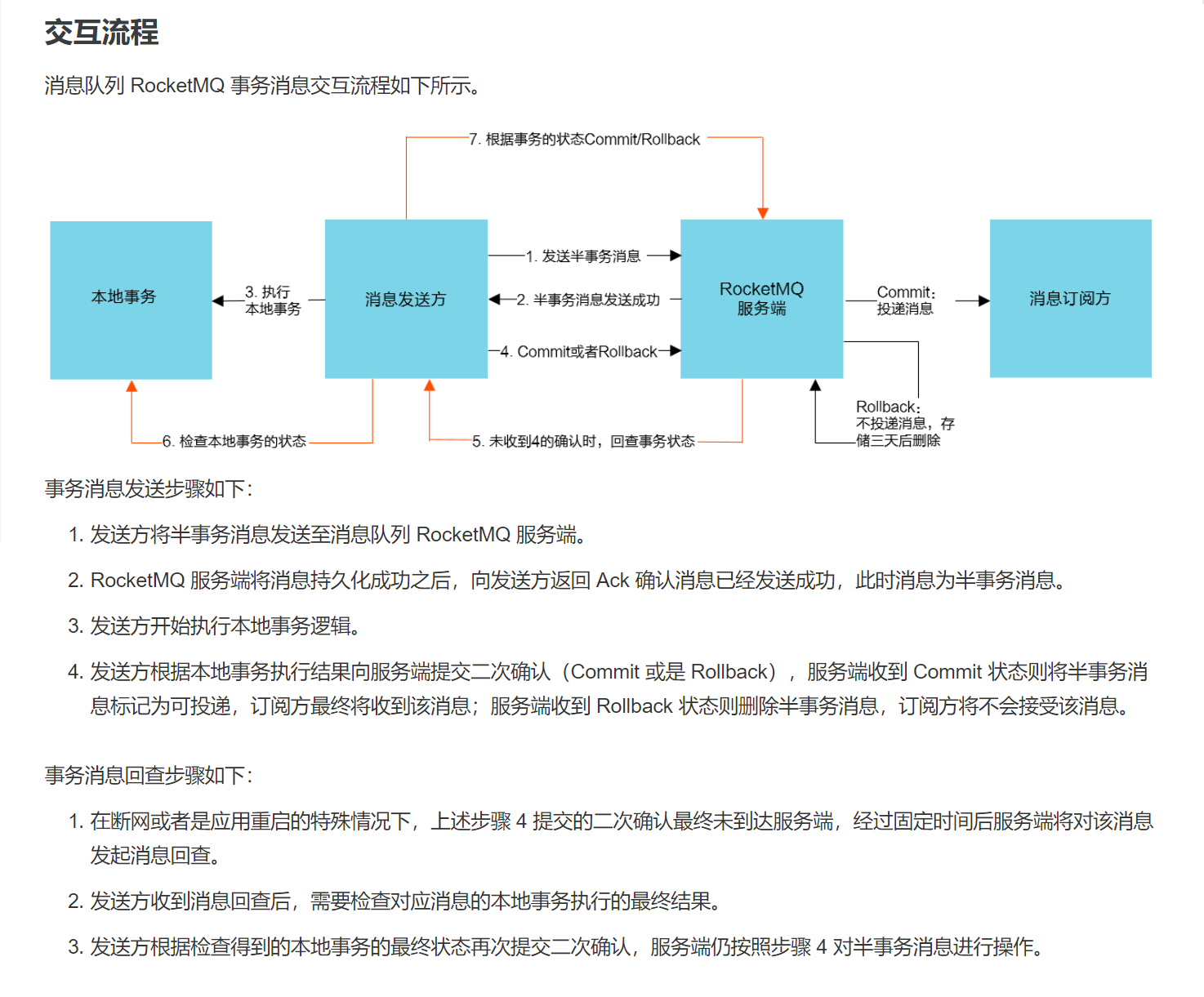
分区顺序消息



## 事务消息

半事务消息：暂不能投递的消息，发送方已经成功地将消息发送到RocketMq服务端，但是服务端未接收生产者对该消息的二次确认，此时消息被标记为暂不能投递状态，处于该状态下的消息称为半事务消息。

消息回查：由于网络中断，生产者重启等原因，导致某条事务的二次确认消息丢失，此时RocketMq服务端扫描发现某条消息处于半事务状态，需要主动向生产者询问该消息的最终状态(commit或者rollback)，该询问过程就是消息回查。



## 消息类型对比



## 发送方式对比



# 消息重试

对于顺序消息，不断的进行重试，间隔是1秒。

对于无序消息，无序消息的重试只针对集群消费方式生效，广播方式不提供重试特性。

# 消息过滤

Tag是Topic的下一级分类，所以可以根据Tag进行消息过滤。

## 方式一

1. consumer.subscribe("MQ\_TOPIC", "\*", **new** MessageListener() {
2. **public** Action **consume**(Message message, ConsumeContext context) {
3. System.out.println(message.getMsgID());
4. **return** Action.CommitMessage;
5. }
6. });

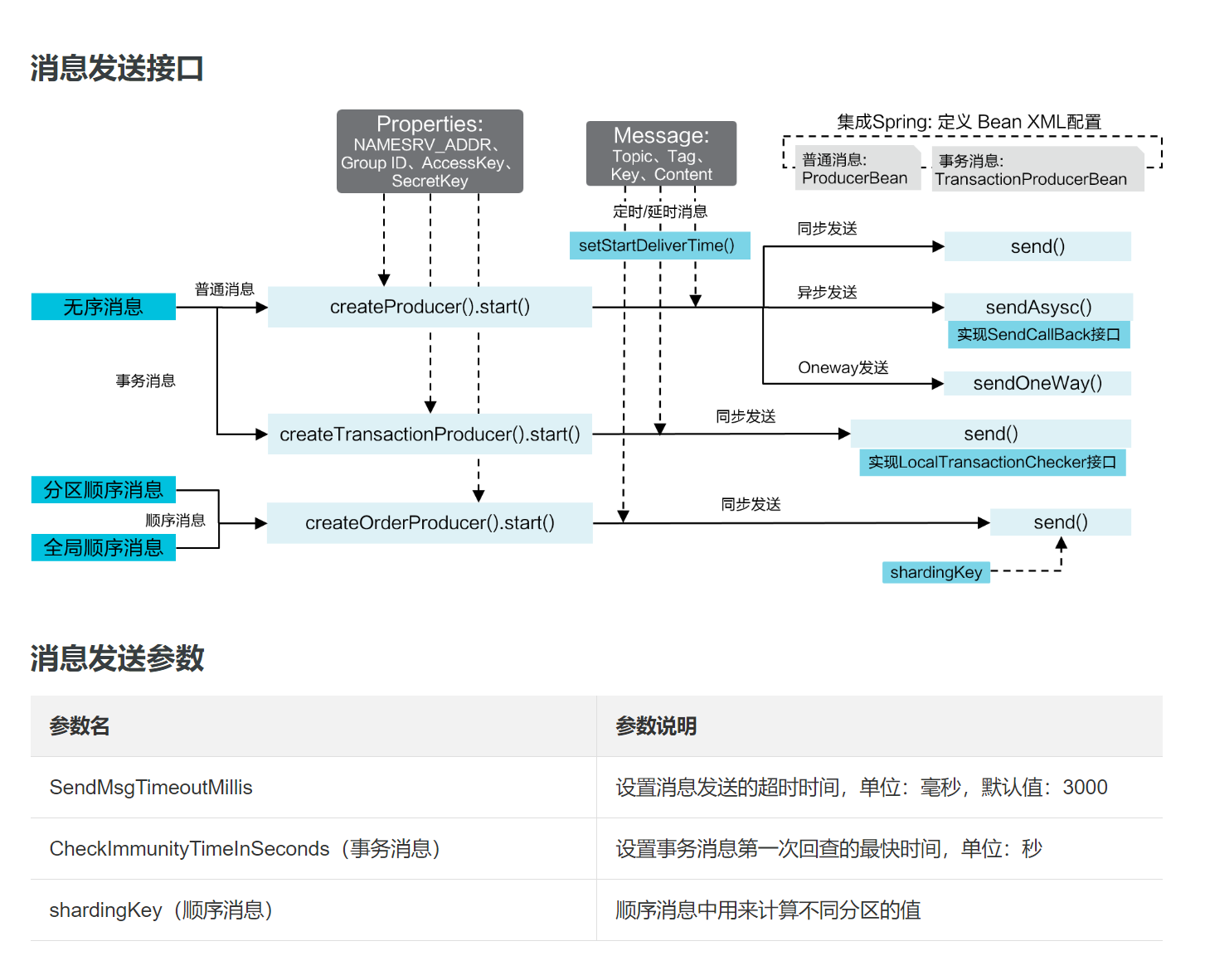
## 方式二

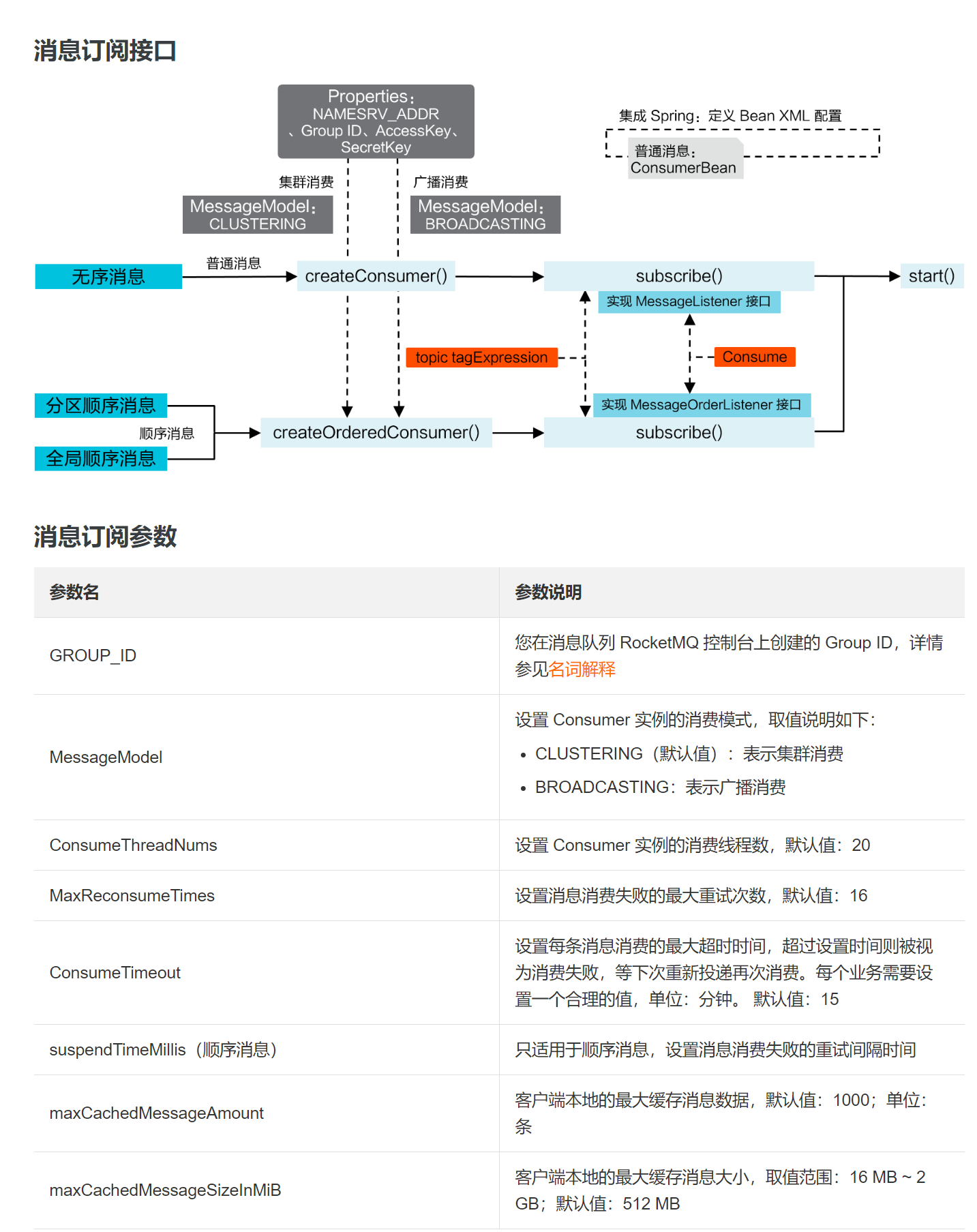
1. consumer.subscribe("MQ\_TOPIC", "TagA", **new** MessageListener() {
2. **public** Action **consume**(Message message, ConsumeContext context) {
3. System.out.println(message.getMsgID());
4. **return** Action.CommitMessage;
5. }
6. });

## 方式三

1. consumer.subscribe("MQ\_TOPIC", "TagA||TagB", **new** MessageListener() {
2. **public** Action **consume**(Message message, ConsumeContext context) {
3. System.out.println(message.getMsgID());
4. **return** Action.CommitMessage;
5. }
6. });

# 接口和参数说明



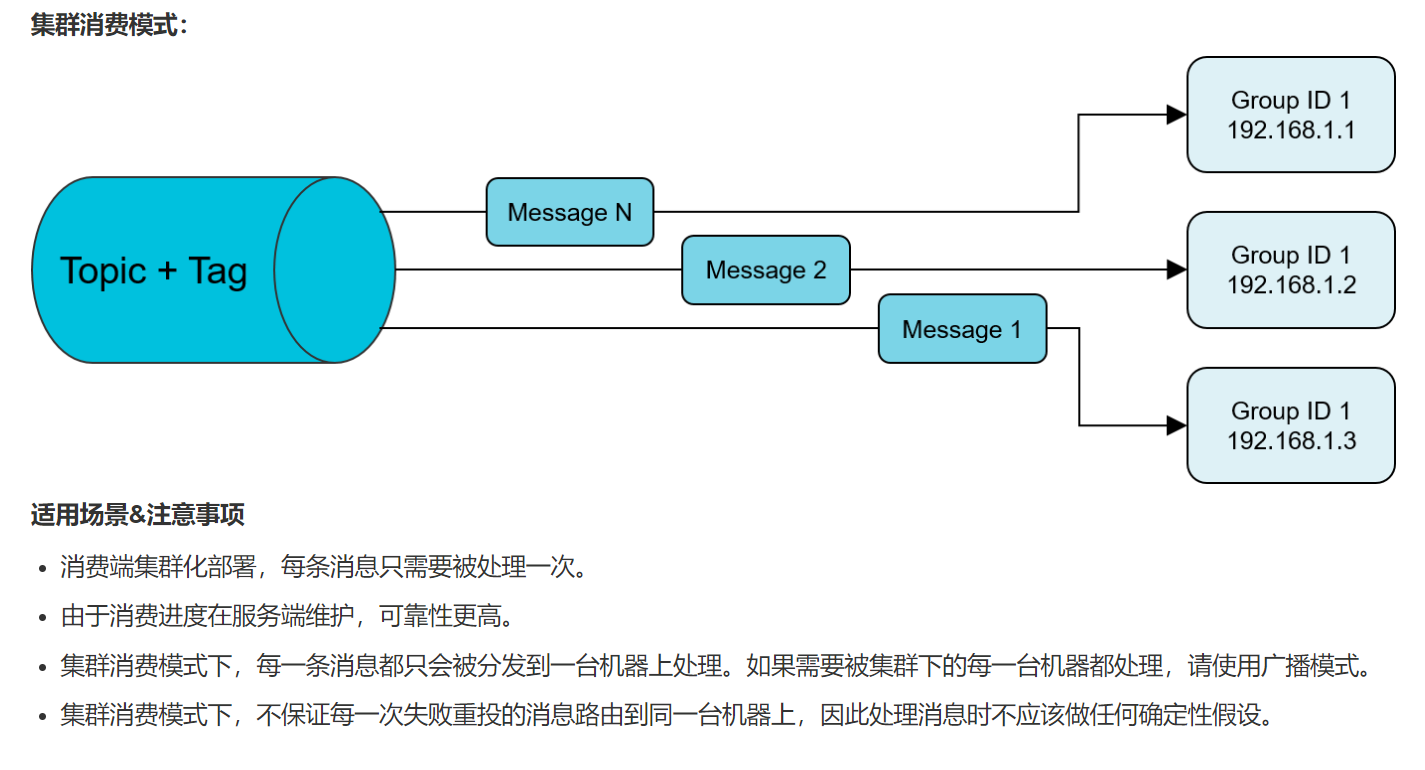


# 消费方式

## 集群消费

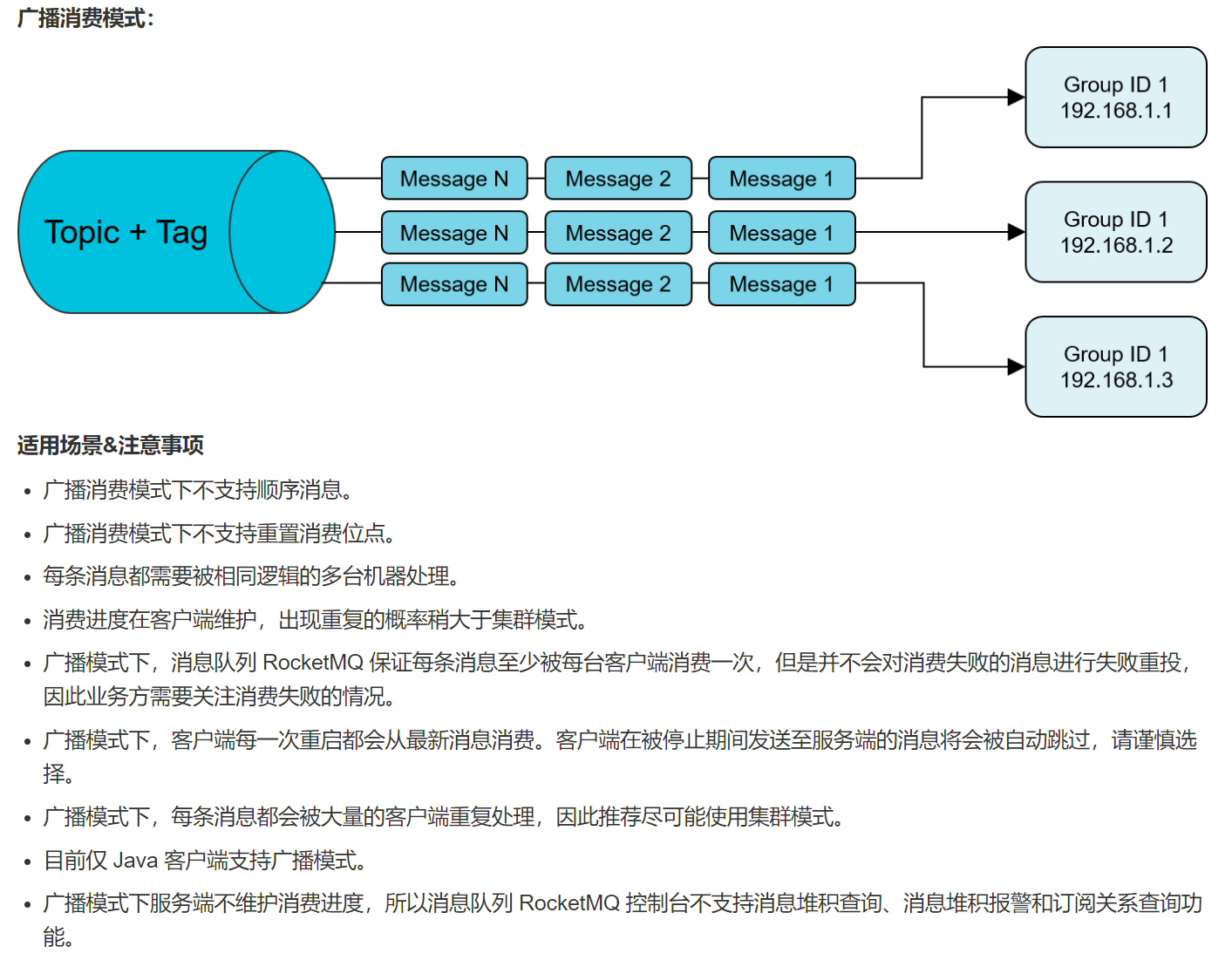
使用相同Group ID的订阅者属于同一个集群，同一个集群下的订阅者消费逻辑必须完全一致，包括Tag的使用。这些订阅者在逻辑上可以认为是一个消费节点。

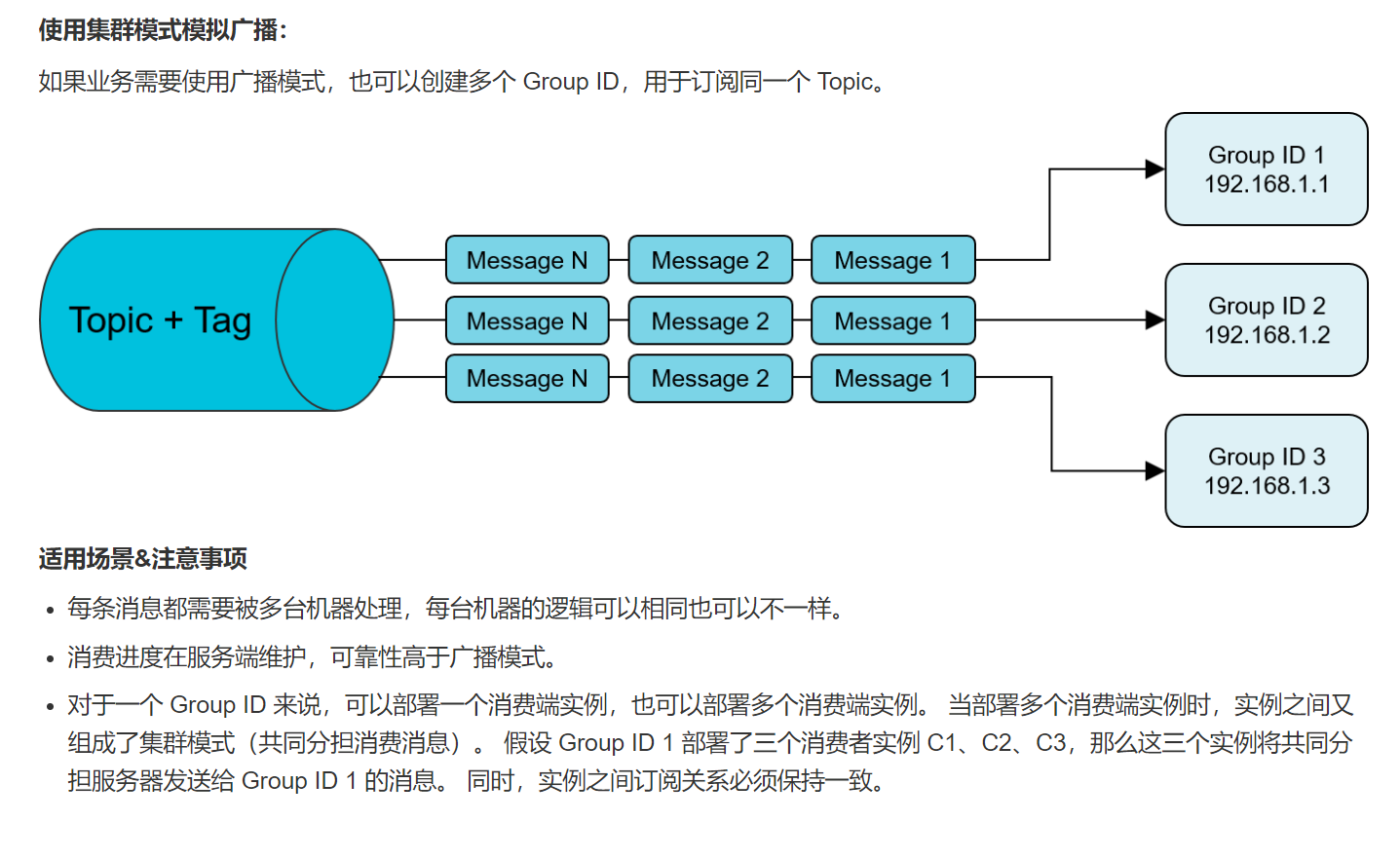
集群消费模式时，任意一条消息只需要被集群内的任意一个消费者处理即可。



## 广播消费

会将每条消息推送给集群内所有注册过的客户端，保证消息至少被每台机器消费一次。





# 收发消息

## 收发顺序消息

## 收发事务消息

## 收发延时消息

## 收发定时消息

# 生产者

消息只有发送bytes类型的，这里是将对象转换成Json串，然后通过Bytes格式传输。

|  |
| --- |
| package rocketmq;  import com.alibaba.fastjson.JSON;  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.common.message.Message;  import com.alibaba.rocketmq.remoting.exception.RemotingException;  import rocketmq.bean.User;  public class Producer {  public static void main(String[] args) throws InterruptedException, MQClientException, RemotingException, MQBrokerException {  //生成Producer  DefaultMQProducer producer = new DefaultMQProducer("producer\_group");  //配置Producer  producer.setNamesrvAddr("172.16.192.50:9876");  //启动Producer  try{  producer.start();  }catch(MQClientException e) {  e.printStackTrace();  return;  }  //生产消息  //String str = "今天星期er";  User user = new User();  for (int i = 1; i <= 100; i++) {  user.setId(i+"");  user.setName("张三");  user.setAge("15");  user.setSex("1");  /\*\*  \* 将对象转为Json-两种方式  \*/  //String str = new Gson().toJson(user);  String str = JSON.toJSONString(user);    Message msg = new Message("test\_topic", "test\_tags",str.getBytes());  producer.send(msg);    Thread.sleep(1000);  }  //停止Producer  producer.shutdown();  }  } |

# 消费者

获取到的数据是Bytes格式的消息，先转换成Json串，然后将Json串转换为对象。

|  |
| --- |
| package rocketmq;  import java.util.List;  import com.alibaba.fastjson.JSON;  import com.alibaba.rocketmq.client.consumer.DefaultMQPushConsumer;  import com.alibaba.rocketmq.client.consumer.listener.ConsumeConcurrentlyContext;  import com.alibaba.rocketmq.client.consumer.listener.ConsumeConcurrentlyStatus;  import com.alibaba.rocketmq.client.consumer.listener.MessageListenerConcurrently;  import com.alibaba.rocketmq.common.message.Message;  import com.alibaba.rocketmq.common.message.MessageExt;  import com.alibaba.rocketmq.common.protocol.heartbeat.MessageModel;  import rocketmq.bean.User;  public class ConsumerTest2 {    public static void main(String[] args) {  DefaultMQPushConsumer consumer=new DefaultMQPushConsumer("consumer\_group");  consumer.setNamesrvAddr("127.0.0.1:9876");  try {    // 订阅PushTopic下Tag为push的消息,都订阅消息  consumer.subscribe("test\_topic", "\*");  //消费模式  consumer.setMessageModel(MessageModel.BROADCASTING);    //可以修改每次消费消息的数量，默认设置是每次消费一条  // consumer.setConsumeMessageBatchMaxSize(10);    //注册消费的监听  consumer.registerMessageListener(new MessageListenerConcurrently() {  //在此监听中消费信息，并返回消费的状态信息  public ConsumeConcurrentlyStatus consumeMessage(  List<MessageExt> msgs,  ConsumeConcurrentlyContext context) {    // msgs中只收集同一个topic，同一个tag，并且key相同的message  // 会把不同的消息分别放置到不同的队列中  for(Message msg:msgs){  String string = new String(msg.getBody());  /\*\*  \* 将Json格式的消息转换成对象  \*/  User user = JSON.parseObject(string,User.class);  System.out.println(user+"=======对象");  //System.out.println(new String(msg.getBody())+"=======Json");  }  return ConsumeConcurrentlyStatus.CONSUME\_SUCCESS;  }  });    consumer.start();    } catch (Exception e) {  e.printStackTrace();  }  }  } |

# 日志配置