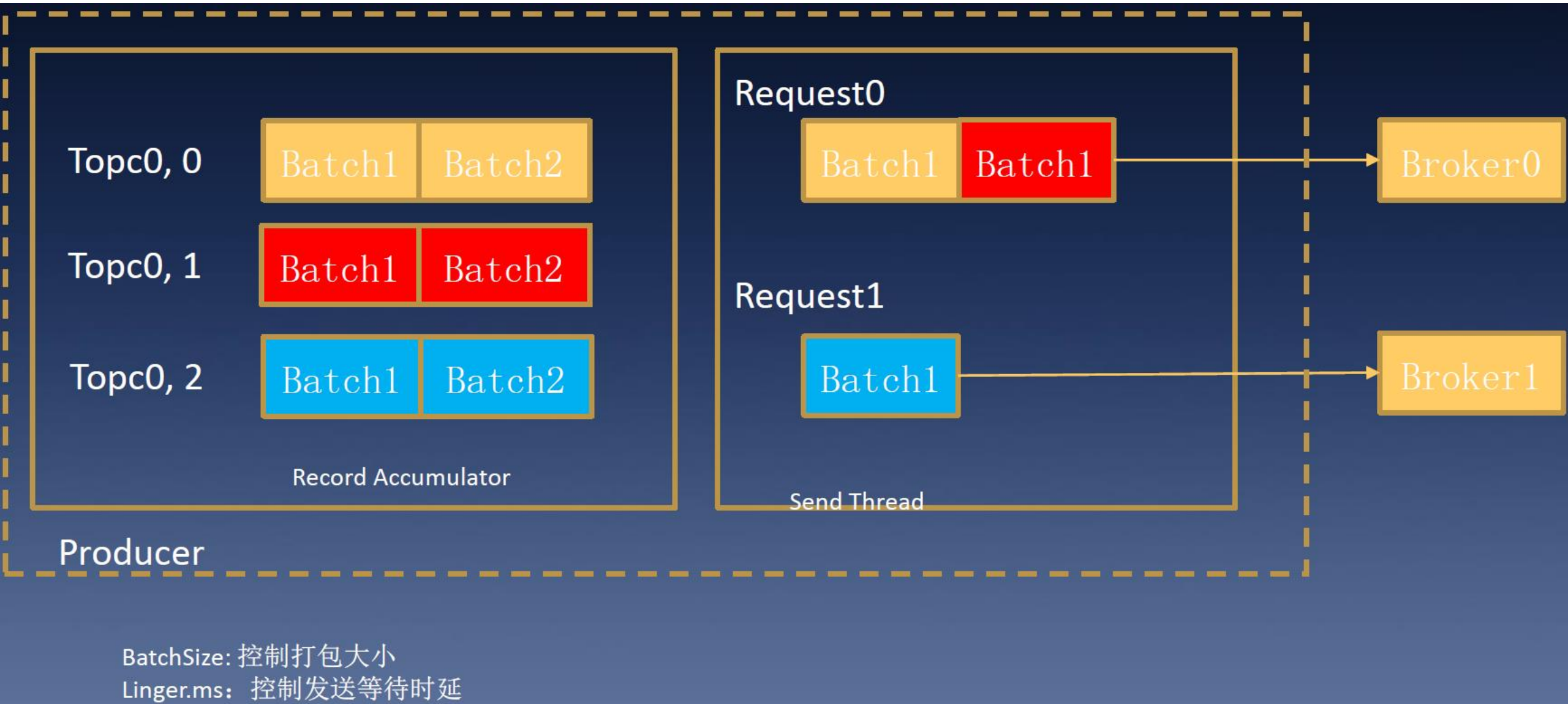
Kafka 课程培训：生产机制

目录

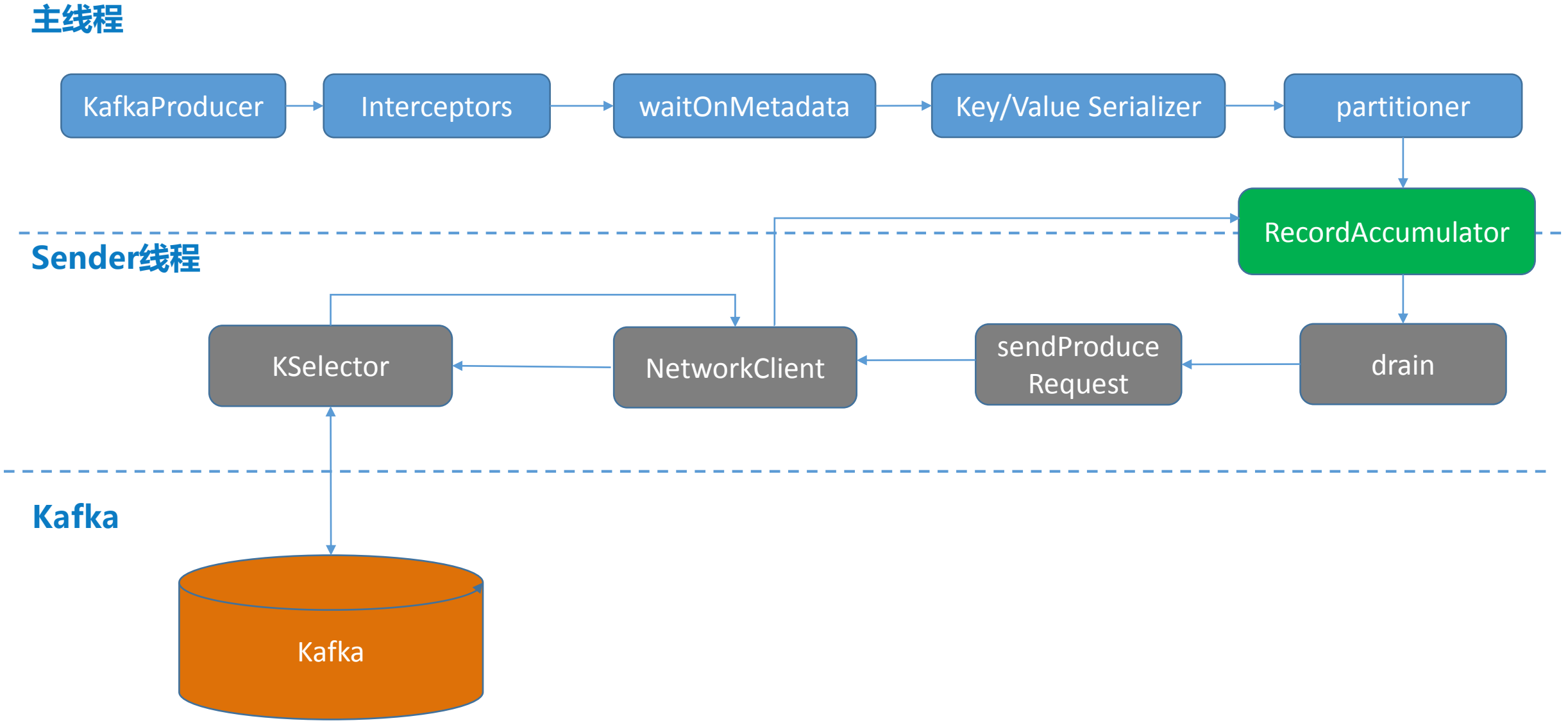
1.1 生产模型



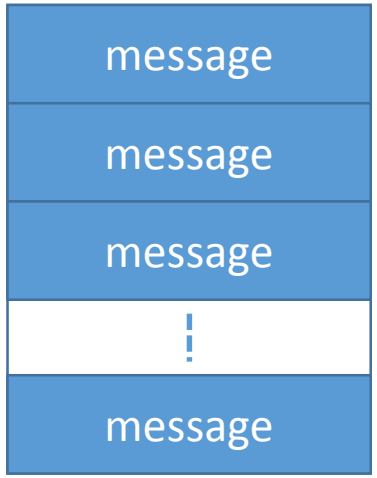
BatchSize: 控制打包大小

Linger.ms： 控制发送等待时延

1.2 生产模型：消息生产流程



1.3 生产模型：Batch



- **batch.size**

默认16384，消息batch大小，当batch达到batch.size大小时，唤醒sender线程发送消息。批量发送消息，减少request数量，提升发送效率，减轻服务端压力

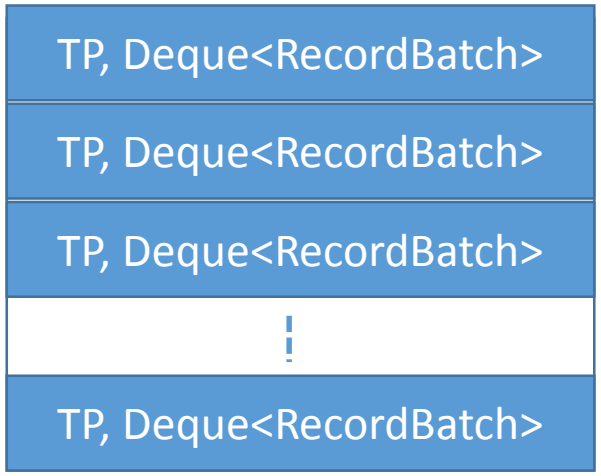
- **ling.ms**

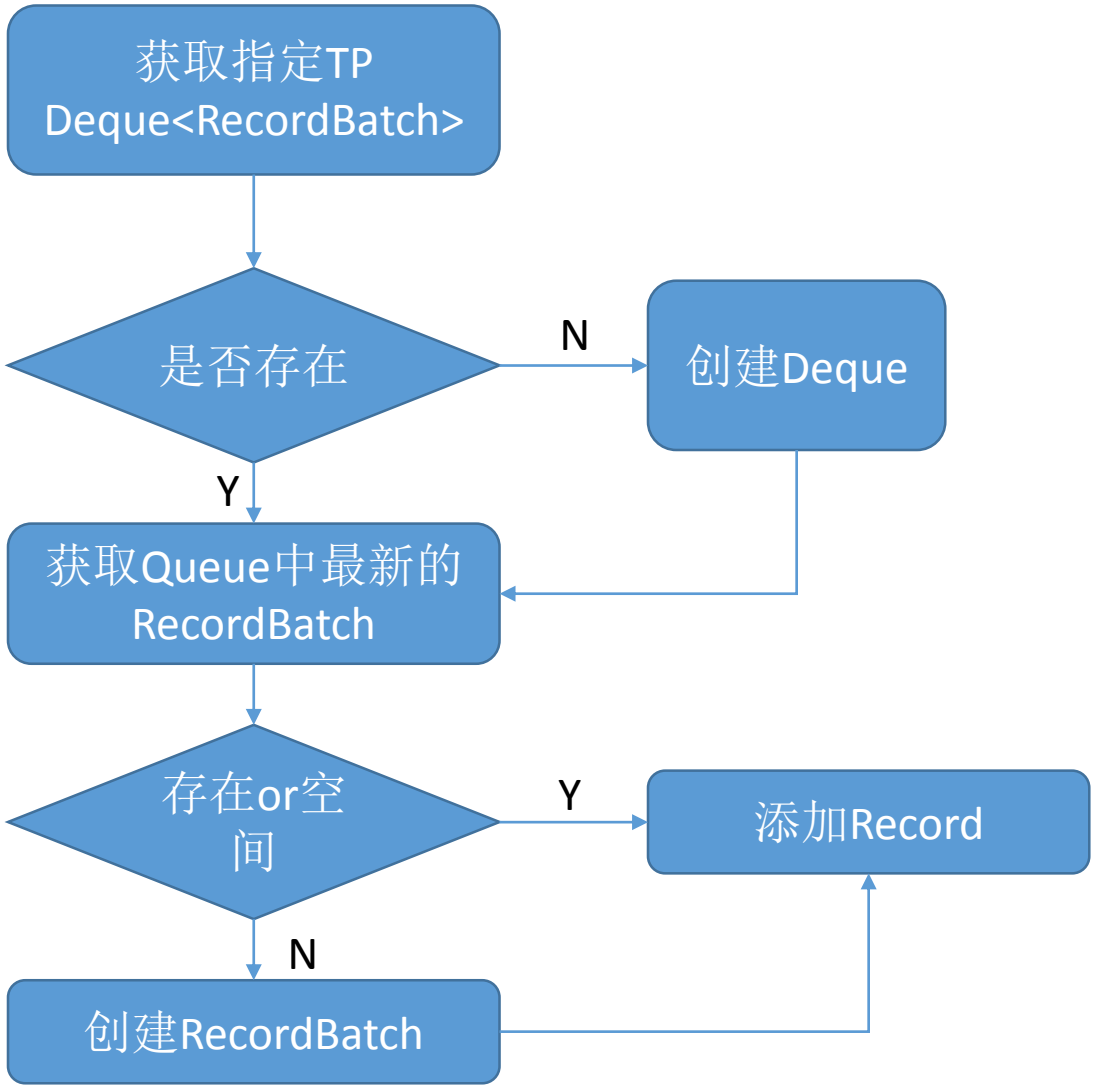
默认0，sender线程检查batch是否ready，满足batch.size和ling.ms其中一个，即发送消息

- **buffer.memory**

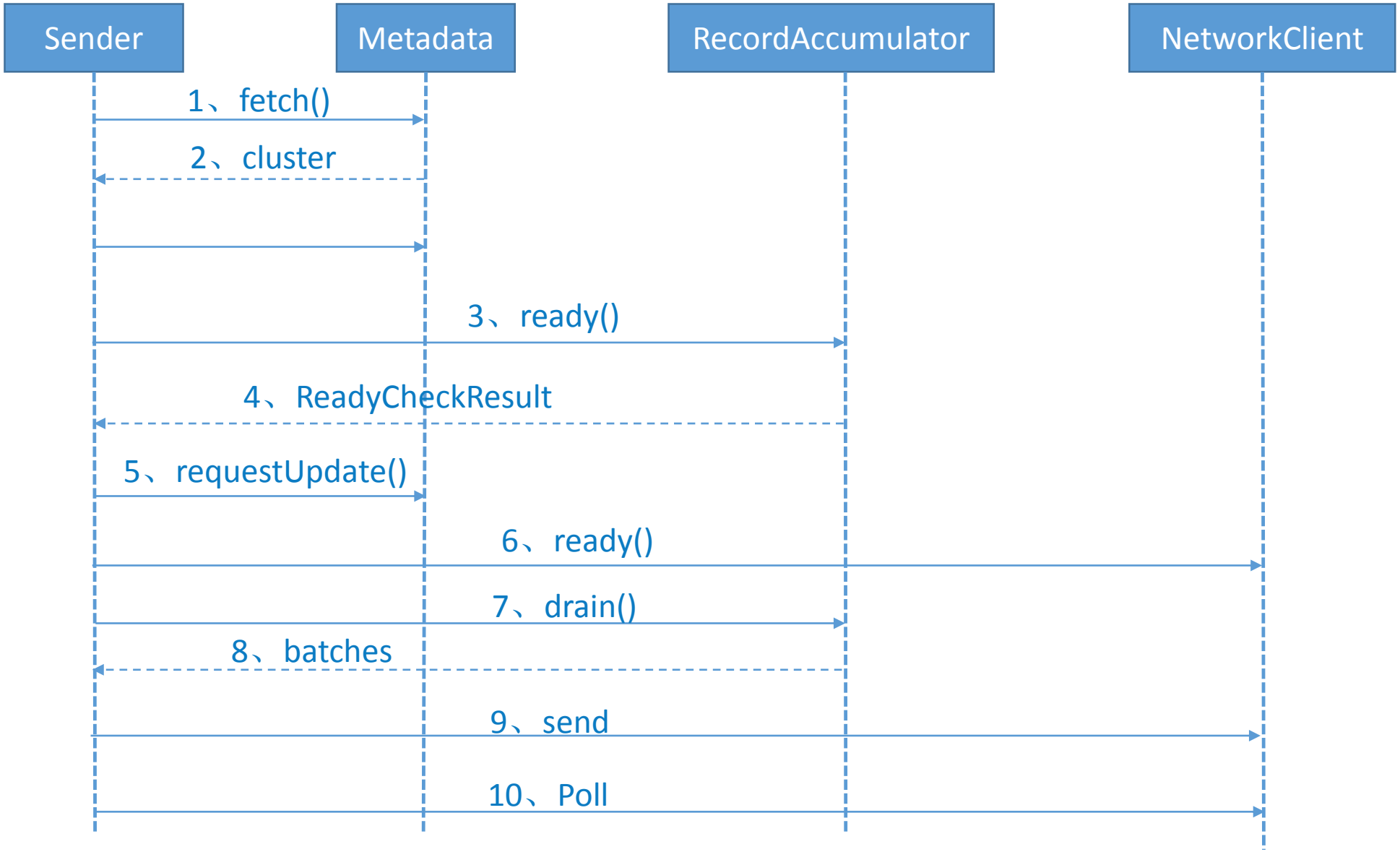
默认33554432，producer可以用来缓存数据的内存大小。如果数据产生速度大于向broker发送的速度，producer会阻塞或者抛出异常，通过参数“block.on.buffer.full”控制

1.3 生产模型：Record Accumulator





1.4 生产模型：Sender Thread



2.1 参数调优

参数	默认值	推荐值	说明
buffer.memory	33554432	536870912	producer可以用来缓存数据的内存大小。如果数据产生速度大于向broker发送的速度，producer会阻塞或者抛出异常，以“block.on.buffer.full”来表明。这项设置将和producer能够使用的总内存相关，但并不是一个硬性的限制，因为不是producer使用的所有内存都是用于缓存。一些额外的内存会用于压缩（如果引入压缩机制），同样还有一些用于维护请求。
linger.ms	0	0	producer组将会汇总任何在请求与发送之间到达的消息记录一个单独批量的请求。通常来说，这只有在记录产生速度大于发送速度的时候才能发生。然而，在某些条件下，客户端将希望降低请求的数量，甚至降低到中等负载一下。这项设置将通过增加小的延迟来完成--即，不是立即发送一条记 录，producer将会等待给定的延迟时间以允许其他消息记录发送，这些消息记录可以批量处理。这可以认为是TCP种Nagle的算法类似。这项设置设 定了批量处理的更高的延迟边界：一旦我们获得某个partition的batch.size，他将会立即发送而不顾这项设置，然而如果我们获得消息字节数 比这项设置要小的多，我们需要“linger”特定的时间以获取更多的消息。 这个设置默认为0，即没有延迟。设定linger.ms=5，例如，将会减少请求数目，但是同时会增加5ms的延迟。
receive.buffer.bytes	32768	默认值或者略大于带宽*时延	TCP receive缓存大小，当读取数据时使用
send.buffer.bytes	131072	默认值或者略大于带宽*时延	TCP send缓存大小，当发送数据时使用

2.2 参数调优

参数	默认值	推荐值	说明
acks	1	1	producer需要server接收到数据之后发出的确认接收的信号，此项配置就是指producer需要多少个这样的确认信号。此配置实际上代表了数据备份的可用性。以下设置为常用选项： （1）acks=0： 设置为0表示producer不需要等待任何确认收到的信息。副本将立即加到socket buffer并认为已经发送。没有任何保障可以保证此种情况下server已经成功接收数据，同时重试配置不会发生作用（因为客户端不知道是否失败）回 馈的offset会总是设置为-1； （2）acks=1： 这意味着至少要等待leader已经成功将数据写入本地log，但是并没有等待所有follower是否成功写入。这种情况下，如果follower没有成功备份数据，而此时leader又挂掉，则消息会丢失。 （3）acks=all： 这意味着leader需要等待所有备份都成功写入日志，这种策略会保证只要有一个备份存活就不会丢失数据。这是最强的保证。 目前Broker端限制只支持这3个值。
batch.size	16384	262144	producer将试图批处理消息记录，以减少请求次数。这将改善client与server之间的性能。这项配置控制默认的批量处理消息字节数。 不会试图处理大于这个字节数的消息字节数。 发送到brokers的请求将包含多个批量处理，其中会包含对每个partition的一个请求。 较小的批量处理数值比较少用，并且可能降低吞吐量（0则会仅用批量处理）。较大的批量处理数值将会浪费更多内存空间，这样就需要分配特定批量处理数值的内存大小。

2.3 建议规范：Producer配置

1.同步复制客户端还需要配合使用：ack=all

2.配置发送失败重试：retries=3

3.发送优化：linger.ms=0

4.生产端的JVM内存要足够，避免内存不足导致发送阻塞

5.Callback函数不能阻塞，否则会阻塞sender线程

2.4 配置场景：FIFO配置，消息保序

1.生产消息指定partiton：

ProducerRecord(String topic, Integer partition, K key, V value)

2.配置发送失败重试为0：

retries=0

3.或者设置请求发送队列长度为1：

max.in.flight.requests.per.connection=1

2.5 配置场景：高吞吐生产配置

1.Topic配置：

3分区、2副本

2.配置发送确认配置：

acks=0 or 1

2.6 配置场景：相对可靠配置

1.Topic配置：

3分区、3副本

min.insync.replicas=2

2.配置发送确认配置：

acks=-1

2.7 配置场景：高可靠配置

1.Topic配置：

- 3分区、3副本
- min.insync.replicas=2
- flush.messages=1

2.配置发送确认配置：

acks=-1

3 代码示例

```
public class KafkaProducerDemo
{
    public static void main(String[] args) throws InterruptedException, ExecutionException
    {
        if (args.length != 2)
        {
            throw new IllegalArgumentException("usage: dms.kafka.demo.KafkaProducerDemo bootstrap-servers topic-name.");
        }
        Properties props = new Properties();
        props.put("bootstrap.servers", args[0]);
        props.put("acks", "all");
        props.put("retries", 0);
        props.put("batch.size", 16384);
        props.put("linger.ms", 1);
        props.put("buffer.memory", 33554432);
        props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

        Producer<String, String> producer = new KafkaProducer<>(props);
        for (int i = 0; i < 100; i++)
        {
            Future<RecordMetadata> result =
                producer.send(new ProducerRecord<String, String>(args[1], Integer.toString(i), Integer.toString(i)));
            RecordMetadata rm = result.get();
            System.out.println("topic: " + rm.topic() + ", partition: " + rm.partition() + ", offset: " + rm.offset());
        }
        producer.close();
    }
}
```

实战演练

- 创建topic：topic-test01
- 编写代码生产消息

Thank You