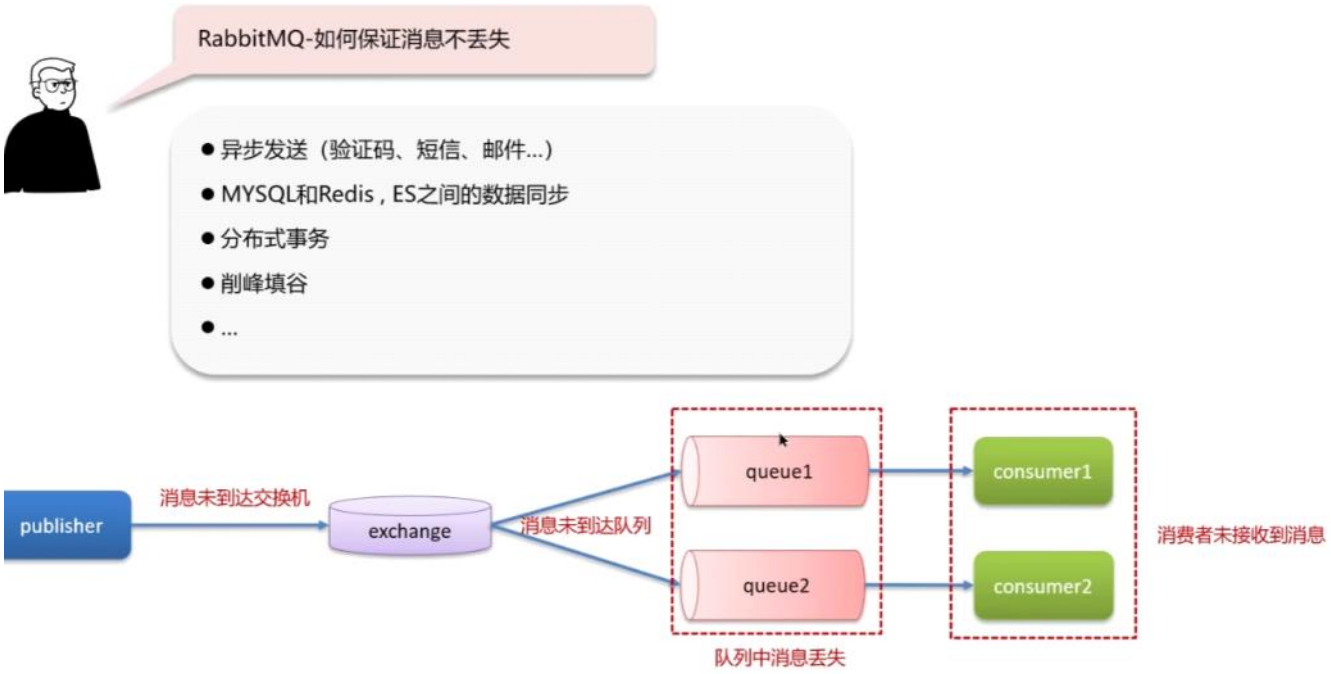
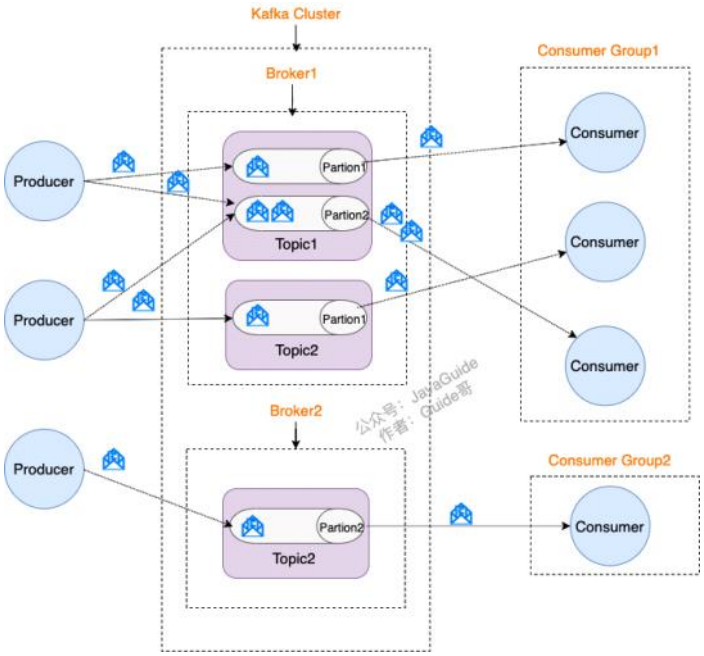
消息中间件Kafka

2024年4月6日 18:15

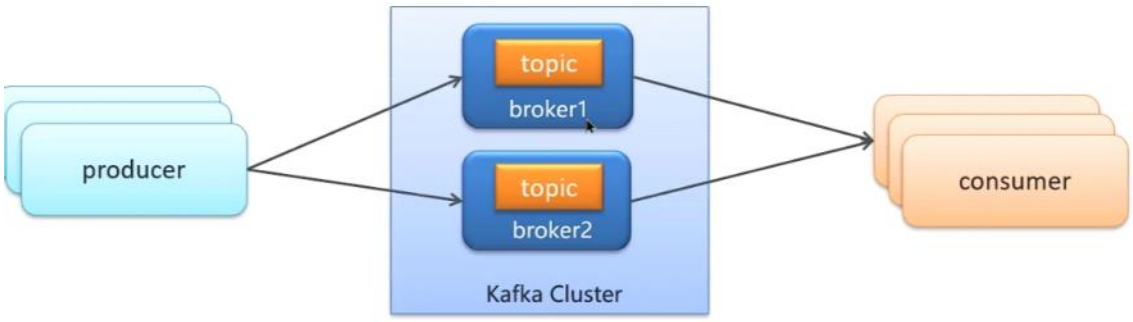




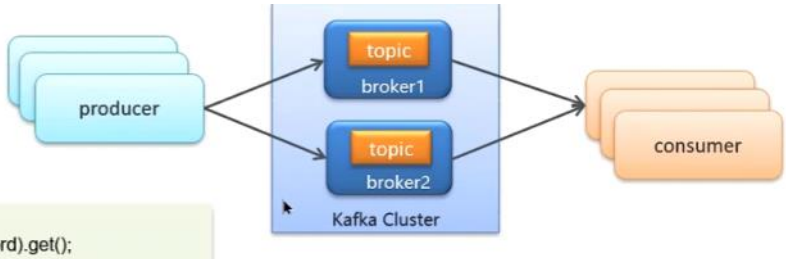
Kafka是如何保证消息不丢失

使用Kafka在消息的收发过程都会出现消息丢失，Kafka分别给出了解决方案

- 生产者发送消息到Brocker丢失
- 消息在Brocker中存储丢失
- 消费者从Brocker接收消息丢失



生产者发送消息到Brocker丢失



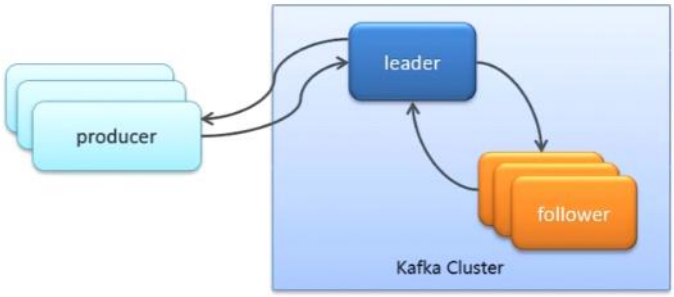
- 设置异步发送

```
//同步发送
RecordMetadata recordMetadata = kafkaProducer.send(record).get();
//异步发送
kafkaProducer.send(record, new Callback() {
    @Override
    public void onCompletion(RecordMetadata recordMetadata, Exception e) {
        if (e != null) {
            System.out.println("消息发送失败 | 记录日志");
        }
        long offset = recordMetadata.offset();
        int partition = recordMetadata.partition();
        String topic = recordMetadata.topic();
    }
});
```

- 消息重试

```
//设置重试次数
prop.put(ProducerConfig.RETRIES_CONFIG,10);
```

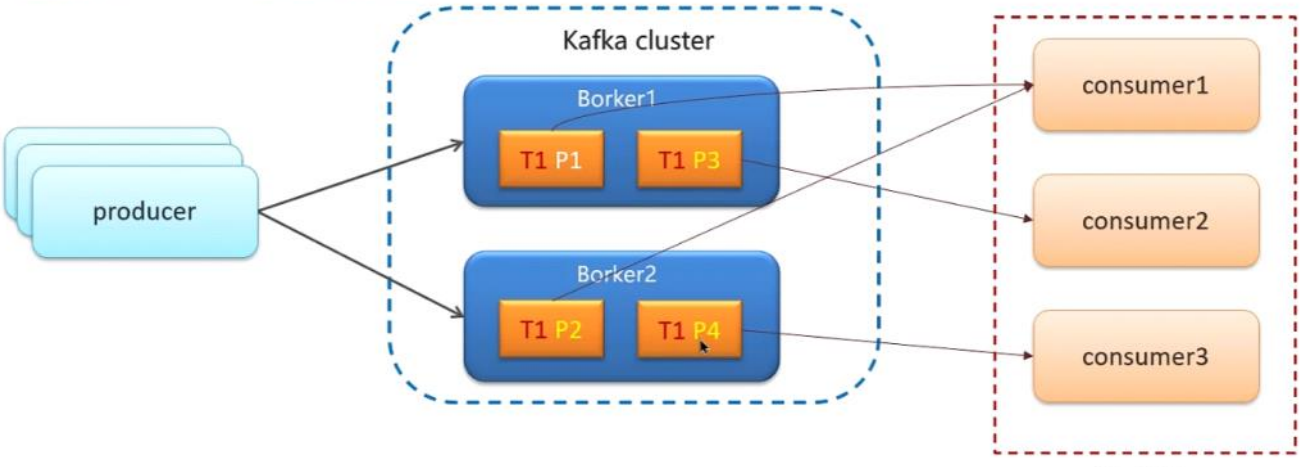
消息在Brocker中存储丢失



- 发送确认机制acks

确认机制	说明
acks=0	生产者在成功写入消息之前不会等待任何来自服务器的响应,消息有丢失的风险，但是速度最快
acks=1（默认值）	只要集群首领节点收到消息，生产者就会收到一个来自服务器的成功响应
acks=all	只有当所有参与赋值的节点全部收到消息时，生产者才会收到一个来自服务器的成功响应

消费者从Brocker接收消息丢失

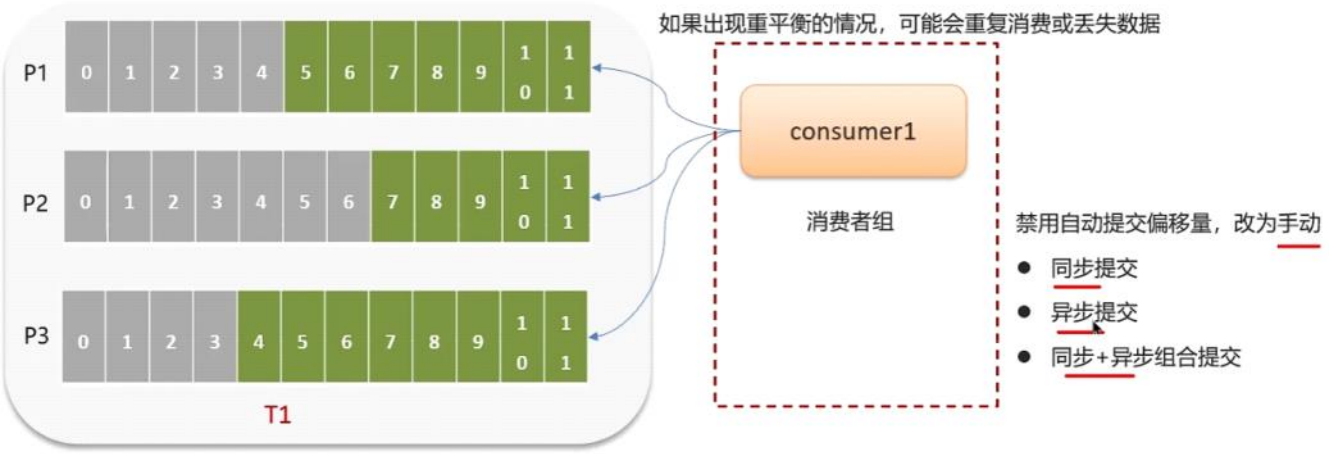


- Kafka 中的分区机制指的是将每个主题划分成多个分区（Partition）
- topic分区中消息只能由消费者组中的唯一一个消费者处理，不同的分区分配给不同的消费者（同一个消费者组）

消费者从Brocker接收消息丢失

消费者默认是自动按期提交已经消费的偏移量，默认是每隔5s提交一次

如果出现重平衡的情况，可能会重复消费或丢失数据



Kafka是如何保证消息不丢失

需要从三个层面去解决这个问题：

- 生产者发送消息到Brocker丢失
  - 设置异步发送，发送失败使用回调进行记录或重发
  - 失败重试，参数配置，可以设置重试次数
- 消息在Brocker中存储丢失

  发送确认acks，选择all，让所有的副本都参与保存数据后确认
- 消费者从Brocker接收消息丢失
  - 关闭自动提交偏移量，开启手动提交偏移量
  - 提交方式，最好是同步+异步提交

Kafka中消息的重复消费问题如何解决的

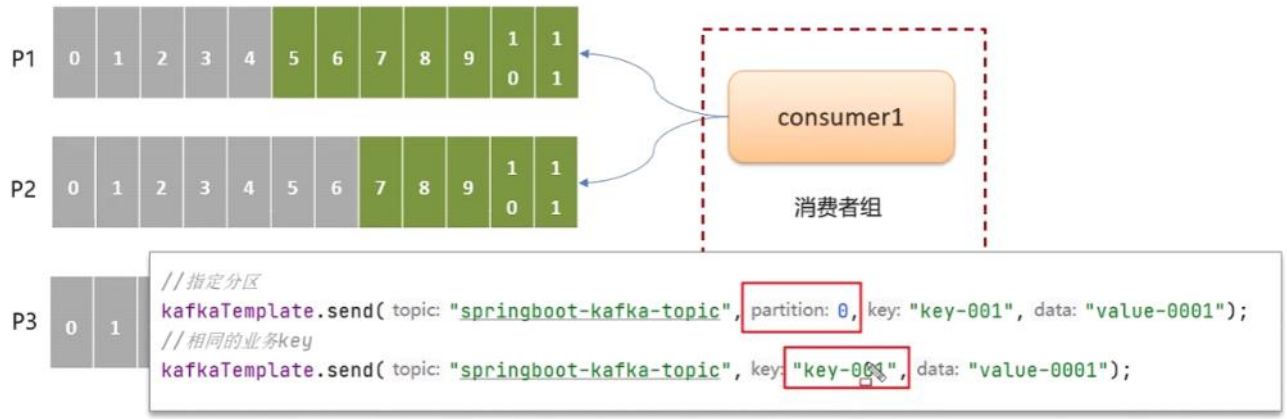
- 关闭自动提交偏移量，开启手动提交偏移量
- 提交方式，最好是同步+异步提交
- 幂等方案

Kafka是如何保证消费的顺序性

应用场景：

- 即时消息中的单对单聊天和群聊，保证发送方消息发送顺序与接收方的顺序一致
- 充值转账两个渠道在同一个时间进行余额变更，短信通知必须要有顺序

消费者从Brocker接收消息丢失



topic分区中消息只能由消费者组中的唯一一个消费者处理，所以消息肯定是按照先后顺序进行处理的。但是它也仅仅是保证Topic的一个分区顺序处理，不能保证跨分区的消息先后处理顺序。 所以，如果你想要顺序的处理Topic的所有消息，那就只提供一个分区。

Kafka是如何保证消费的顺序性

问题原因：

一个topic的数据可能存储在不同的分区中，每个分区都有一个按照顺序的存储的偏移量，如果消费者关联了多个分区不能保证顺序性

解决方案：

- 发送消息时指定分区号
- 发送消息时按照相同的业务设置相同的key

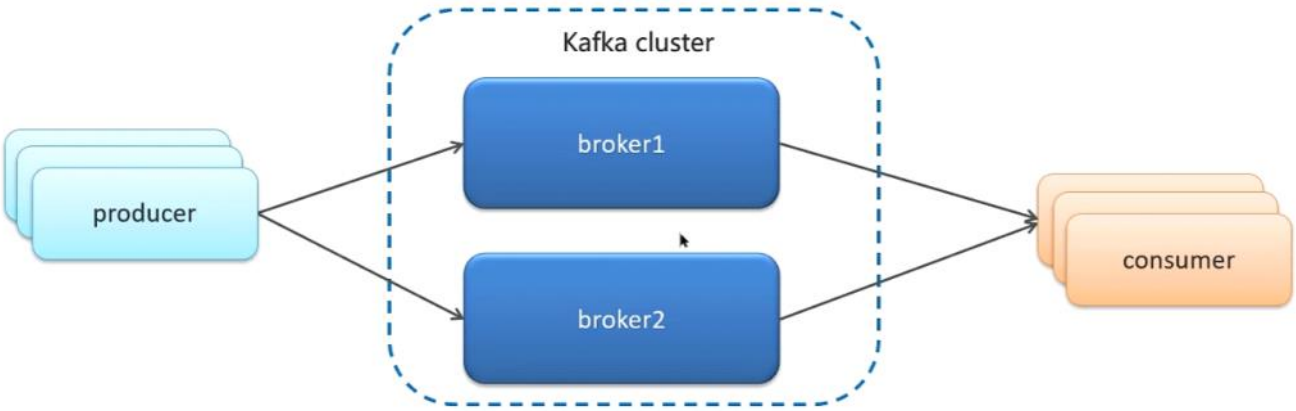
面试官：Kafka是如何保证消费的顺序性

候选人：

kafka默认存储和消费消息，是不能保证顺序性的，因为一个topic数据可能存储在不同的分区中，每个分区都有一个按照顺序的存储的偏移量，如果消费者关联了多个分区不能保证顺序性

如果有这样的需求的话，我们是可以解决的，把消息都存储同一个分区下就行了，有两种方式都可以进行设置，第一个是发送消息时指定分区号，第二个是发送消息时按照相同的业务设置相同的key，因为默认情况下分区也是通过key的hashcode值来选择分区的，hash值如果一样的话，分区肯定也是一样的

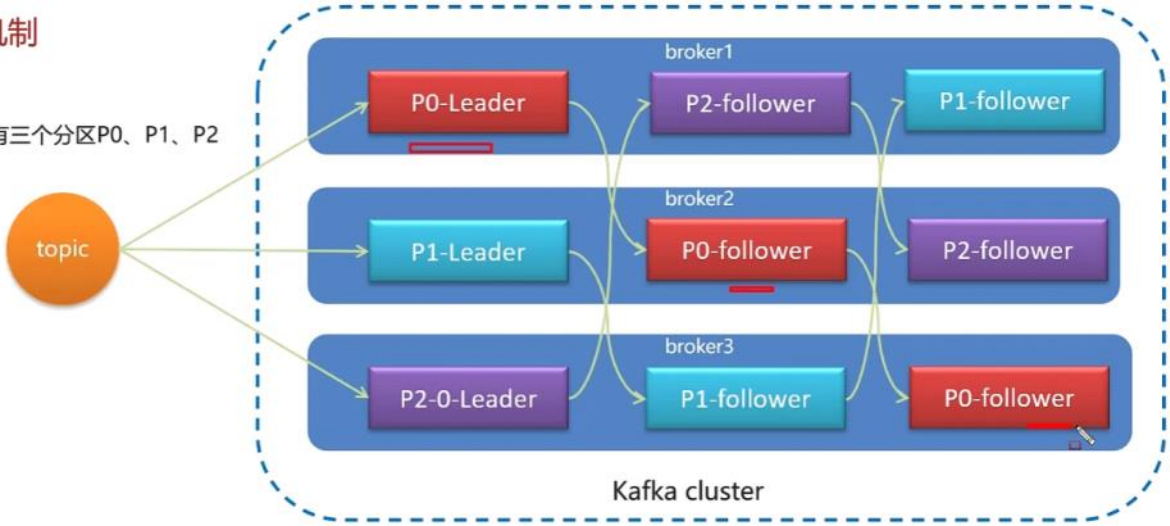
集群模式



- Kafka 的服务器端由被称为 Broker 的服务进程构成，即一个 Kafka 集群由多个 Broker 组成
- 这样如果集群中某一台机器宕机，其他机器上的 Broker 也依然能够对外提供服务。这其实就是 Kafka 提供高可用的手段之一

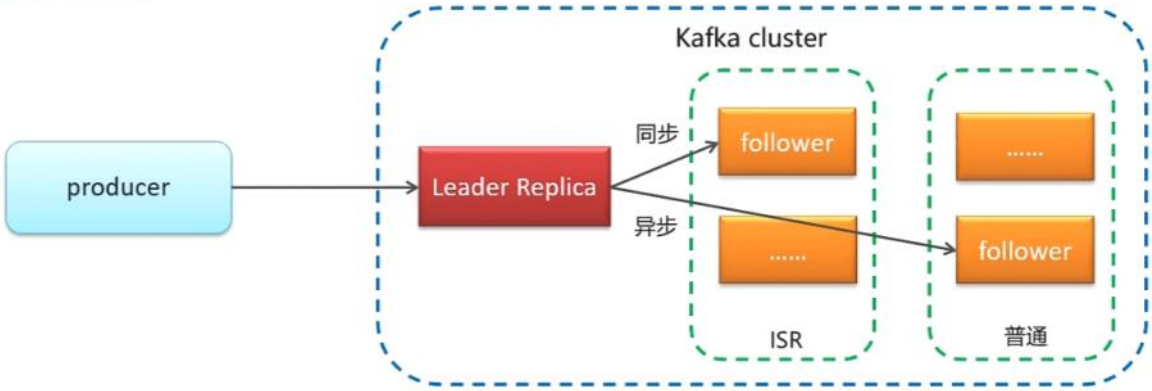
分区备份机制

某一个topic中有三个分区P0、P1、P2



- 一个topic有多个分区，每个分区有多个副本，其中有一个leader，其余的是follower，副本存储在不同的broker中
- 所有的分区副本的内容是都是相同的，如果leader发生故障时，会自动将其中一个follower提升为leader

分区备份机制



ISR（in-sync replica）需要同步复制保存的follower

如果leader失效后，需要选出新的leader，选举的原则如下：

```
//一个topic默认分区的replication个数，不能大于集群中broker的个数，默认为1
default.replication.factor=3
//最小的ISR副本个数
min.insync.replicas=2
```

第一：选举时优先从ISR中选定，因为这个列表中follower的数据是与leader同步的

第二：如果ISR列表中的follower都不行了，就只能从其他follower中选取

Kafka的高可用机制有了解过嘛

可以从两个层面回答，第一个是集群，第二个是复制机制

集群：

一个kafka集群由多个broker实例组成，即使某一台宕机，也不耽误其他broker继续对外提供服务

复制机制：

- 一个topic有多个分区，每个分区有多个副本，有一个leader，其余的是follower，副本存储在不同的broker中
- 所有的分区副本的内容是都是相同的，如果leader发生故障时，会自动将其中一个follower提升为leader，保证了系统的容错性、高可用性

解释一下复制机制中的ISR

ISR（in-sync replica）需要同步复制保存的follower

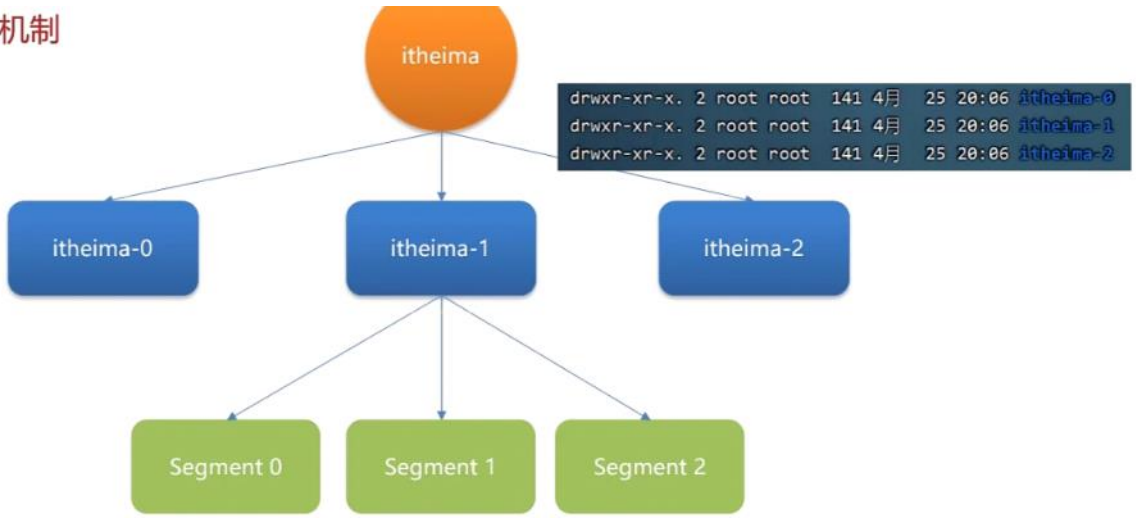
分区副本分为了两类，一个是ISR，与leader副本同步保存数据，另外一个普通的副本，是异步同步数据，当leader挂掉之后，会优先从ISR副本列表中选取一个作为leader

Kafka数据清理机制了解过嘛

- Kafka文件存储机制
- 数据清理机制

Kafka文件存储机制

存储结构：



为什么要分段？

- 删除无用文件方便，提高磁盘利用率
- 查找数据便捷

```
00000000000000000000.index
00000000000000000000.log
00000000000000000000.timeindex
00000000000003987239.index
00000000000003987239.log
00000000000003987239.timeindex
```

.index 索引文件
.log 数据文件
.timeindex 时间索引文件

数据清理机制

日志的清理策略有两个

1. 根据消息的保留时间，当消息在kafka中保存的时间超过了指定的时间，就会触发清理过程

```
# The minimum age of a log file to be eligible for deletion due to age
log.retention.hours=168
```

2. 根据topic存储的数据大小，当topic所占的日志文件大小大于一定的阈值，则开始删除最久的消息。需手动开启

```
# A size-based retention policy for logs. Segments are pruned from the log unless the remaining
# segments drop below log.retention.bytes. Functions independently of log.retention.hours.
#log.retention.bytes=1073741824
```

Kafka数据清理机制了解过嘛

Kafka存储结构

- Kafka中topic的数据存储在分区上，分区如果文件过大会分段存储segment
- 每个分段都在磁盘上以索引(xxxx.index)和日志文件(xxxx.log)的形式存储
- 分段的好处是，第一能够减少单个文件内容的大小，查找数据方便，第二方便kafka进行日志清理。

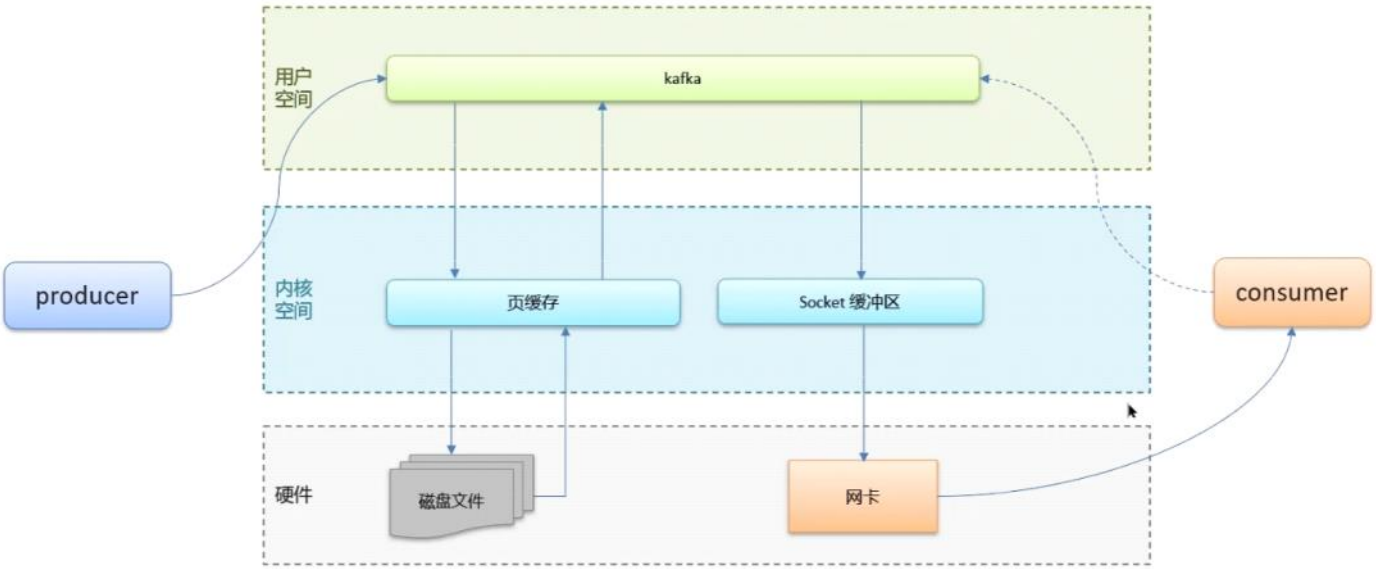
日志的清理策略有两个：

- 根据消息的保留时间，当消息保存的时间超过了指定的时间，就会触发清理，默认是168小时（7天）
- 根据topic存储的数据大小，当topic所占的日志文件大小大于一定的阈值，则开始删除最久的消息。（默认关闭）

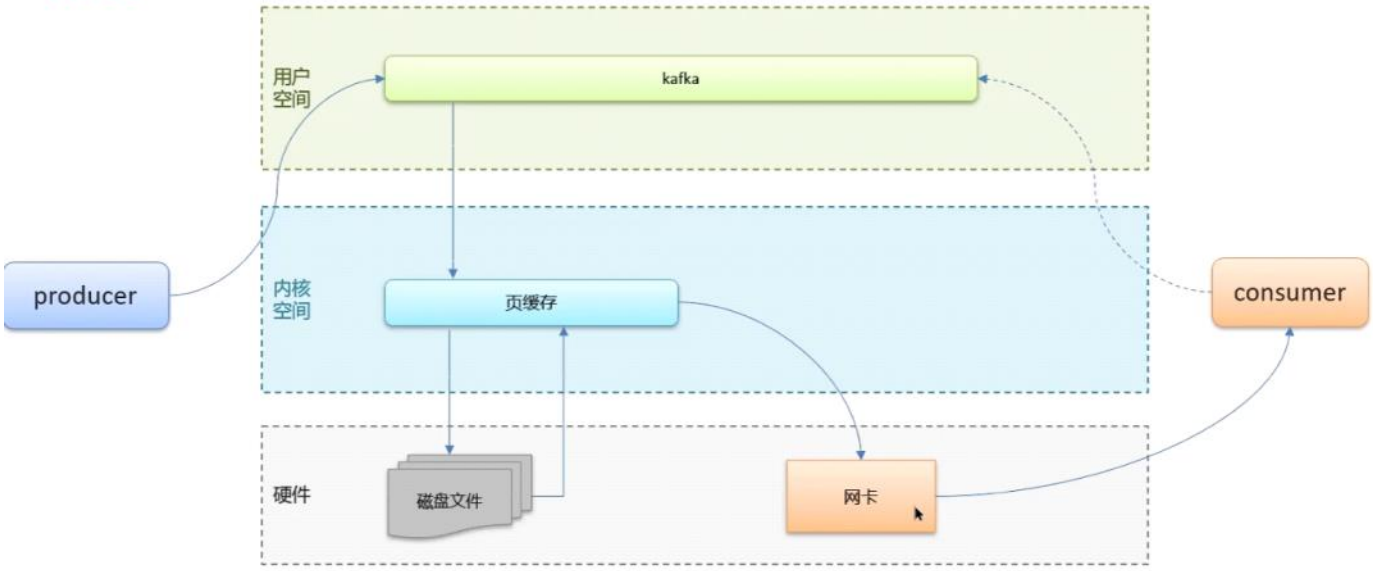
Kafka中实现高性能的设计有了解过嘛

- 消息分区：不受单台服务器的限制，可以不受限的处理更多的数据
- 顺序读写：磁盘顺序读写，提升读写效率
- 页缓存：把磁盘中的数据缓存到内存中，把对磁盘的访问变为对内存的访问
- 零拷贝：减少上下文切换及数据拷贝
- 消息压缩：减少磁盘IO和网络IO
- 分批发送：将消息打包批量发送，减少网络开销

零拷贝



零拷贝



Kafka中实现高性能的设计有了解过嘛

- 消息分区：不受单台服务器的限制，可以不受限的处理更多的数据
- 顺序读写：磁盘顺序读写，提升读写效率
- 页缓存：把磁盘中的数据缓存到内存中，把对磁盘的访问变为对内存的访问
- 零拷贝：减少上下文切换及数据拷贝
- 消息压缩：减少磁盘IO和网络IO
- 分批发送：将消息打包批量发送，减少网络开销