Kafka9 生产者

分区策略

分区原因

1. 方便在集群中扩展，每个Partition可以通过调整以适应它所在的机器，而一个topic又可以有多个Partition组成，因此整个集群就可以适应任意大小的数据了。（负载均衡）
2. 可以提高并发，因为可以以 Partition为单位读写了

分区原则

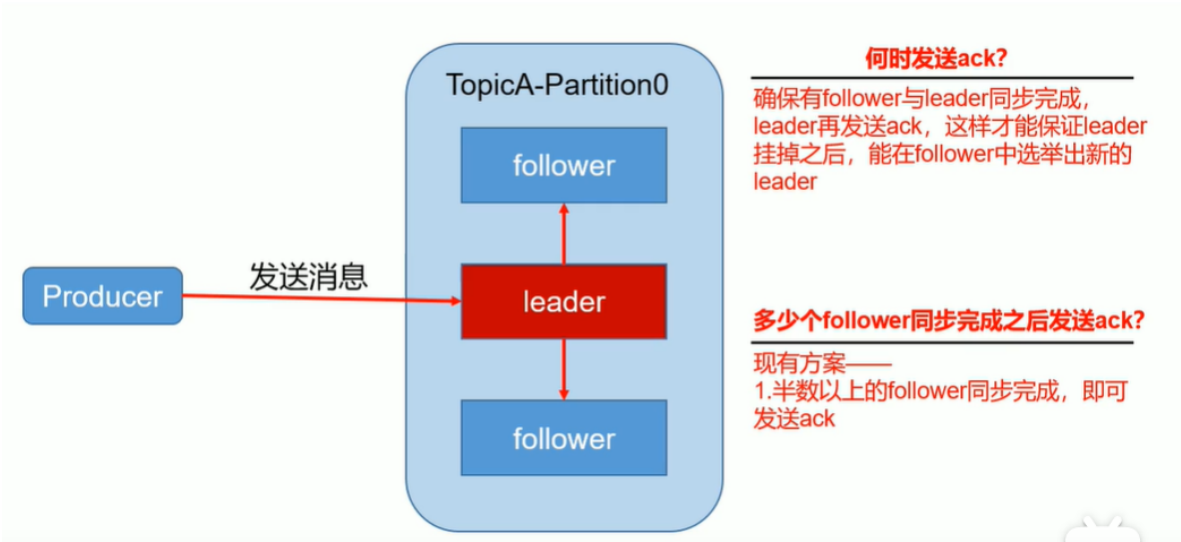
我们需要将 producer 发送的数据封装成一个 ProducerRedcord对象。

```
ProducerRecord(@NotNull String topic, Integer partition, Long timestamp, String key, String value, @Nullable Iterable<Header> headers)
ProducerRecord(@NotNull String topic, Integer partition, Long timestamp, String key, String value)
ProducerRecord(@NotNull String topic, Integer partition, String key, String value, @Nullable Iterable<Header> headers)
ProducerRecord(@NotNull String topic, Integer partition, String key, String value)
ProducerRecord(@NotNull String topic, String key, String value)
ProducerRecord(@NotNull String topic, String value)
```

1. 指明 partition 的情况下，直接将指明的值直接作为partition值
2. 没有指明partition值但是有key的情况下，将key的hash值与topic的partition数进行取余得到partition
3. 既没有partition 值又没有key值的情况下，第一次调用时随机生成一个整数（后面每次调用在这个整数上自增），将这个值与topic可用的partition总数除余得到partition值，也就是常说的round-robin算法

数据可靠性保证

为保证 `producer` 发送的数据，能可靠的发送到指定的 `topic`，`topic` 的每个 `partition` 收到 `producer` 发送的数据后，都需要向 `producer` 发送 `ack`（`acknowledgement` 确认收到），如果 `producer` 收到 `ack`，就会进行下一轮的发送，否则重新发送数据。



副本数据同步策略

方案	优点	缺点
半数以上完成同步	延迟低	选举新的leader时，容忍n台节点的故障，需要2n+1个副本
全部完成同步，才发送ack	选举新的leader时，容忍n台节点的故障，需要n+1个副本	延迟高

Kafka选择了第二种方案，原因如下：

1. 同样为了容忍n台节点的故障，第一种方案需要 2n+1 个副本，而第二种方案只需要 n+1 个副本，而 Kafka 的每个分区都有大量的数据，第一种方案会造成大数据的冗余
2. 虽然第二种网络延迟会比较高，但网络延迟对Kafka的影响较小

ISR

> 背景：采用第二种方案后，设想以下情景：leader收到数据，所有follower都开始同步数据，但有一个follower，因为某种故障，迟迟不能与leader进行同步，那leader就要一直等下去，直到它完成同步，才能发送ack。

Leader 维护了一个动态的 in-sync.replica set（ISR），意为和leader保持同步的follower集合，当ISR中的follower完成数据的同步之后，leader就会给follower发送ack，如果follower长时间未向leader同步数据，则该follower将被踢出ISR，该时间阈值由 replica.lag.time.max.ms 参数设定。Leader发送故障之后，就会从ISR中选举新的leader

ack 应答机制

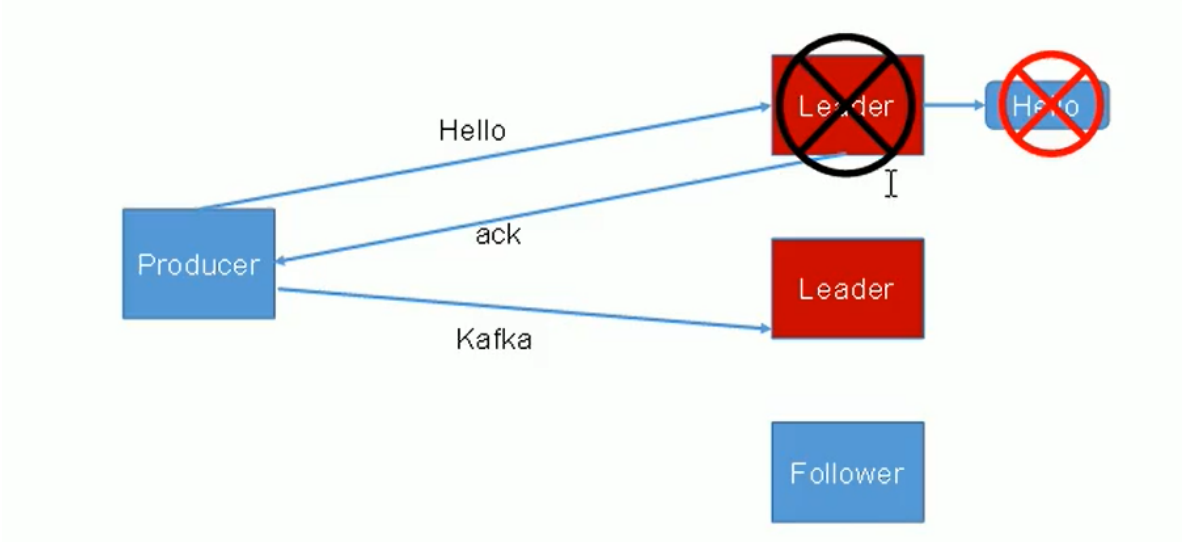
对于某些不太重要的数据，对数据的可靠性要求不是很高，能够容忍数据的少量丢失，所以没必要等ISR中的follower全部接收成功。

所以Kafka为用户提供了三种可靠性级别，用户根据对可靠性和延迟的要求进行权衡，选择以下的配置：

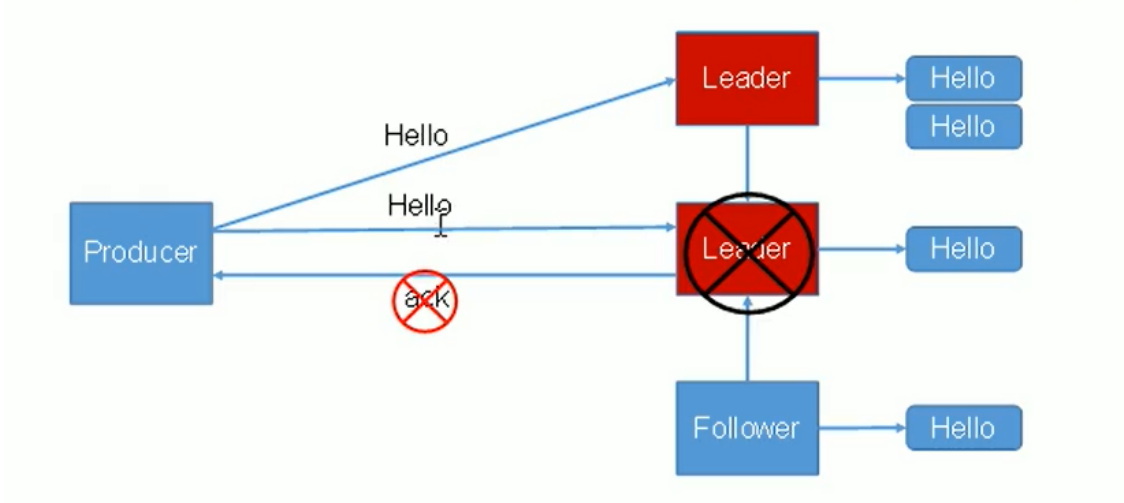
acks参数配置

0: producer 不等待broker的ack，这一操作提供了一个最低的延迟，broker一接收到还没有写入磁盘就已经返回，当broker故障的时候有可能丢失数据
1: prdoucer 等到 broker 的ack，partition的leader落盘成功后返回 ack，如果在follower同步成功之前leader故障，那么将会丢失数据。
-1(all): producer 等待 broker 的 ack， partition 的 leader 和 follower 全部落盘成功后才返回ack。但是如果follower同步完成之后，broker 发送 ack 之前，leader 发生故障，那么会造成数据重复

1. acks= 1 数据丢失案例



2. acks = -1 重复数据案例



故障处理细节


```
LEO：指的是每个副本最大的offset
HW：指的是消费者能见到的最大的offset，ISR队列中最小的LEO（保证消费者的数据一致性）
```

故障处理：

1. follwer 故障：
    follower 发生故障后会被临时踢出ISR，待该follower恢复后，follower会读取本地磁盘记录的上次的HW，并将log文件高于HW的部分截取掉，从HW开始向leader进行同步，等该follower的LEO大于等于该Partition的HW，即follower追上leader之后，就可以重新加入ISR了
2. leader 故障：
    leader发生故障之后，会从ISR中选出一个新的leader，之后，为保证多个副本之间的数据一致性，其余follower会先将各自的log文件高于HW的部分截掉，然后从新的leader同步数据
注意： 这只能保证副本之间的数据一致性，并不能保证数据不丢失或不重复

Exactly Once 语义

将服务器的ACK级别设置为-1，可以不保证Producer到Server之间不会丢失数据，即 At Least Once 语义。相对的，将服务器 ACK 级别设为0， 可以保证生产者每条消息只会发生一次，即 **At Most Once 语义**

At Least Once 可以保证数据不丢失，但是不能保证数据不重复；相对的， At Least Once可以保证数据不重复，但是不能保证数据不丢失。**但是，对于一些非常重要的信息，比如说交易数据，下游数据消费者要求数据既不重复也不丢失，即 Exactly Once语义。**在 0.11版本之前的Kafka，对此是无能为力的。

0.11的版本的Kafka，引入了一个重大特征：幂等性。所谓的幂等性就是指Producer不论向Server发送多少次重复数据，Server端都只会持久化一条。幂等性结合At Least Once 语义，就构成了Kafka的Exactly Once 语义。即：

$$At\ Least\ Once + 幂等性 = Exactly\ Once$$

要启用幂等性，只需要将Producer的参数中 enable.idompotence 设置为true即可，Kafka的幂等性实现其实就是将原来下游需要做的去重放在了数据上游。开启幂等性的Producer在初始化的时候会被分配一个PID，发往同一个 Partition的消息会附带 Sequence Number。而 Broker端只会对<PID，Partition， SeqNumber>做缓存，但具有相同主键的消息提交的时候，Broker只会持久化一条。

但是PID重启就会发生变化，同时不同的Partition也具有不同的主键，所以幂等性无法保证跨分区跨对话的Exactly Once。