> 参考材料：

https://www.bilibili.com/video/BV1a4411B7V9?from=search&seid=373116801260941011

一、Kafka概述

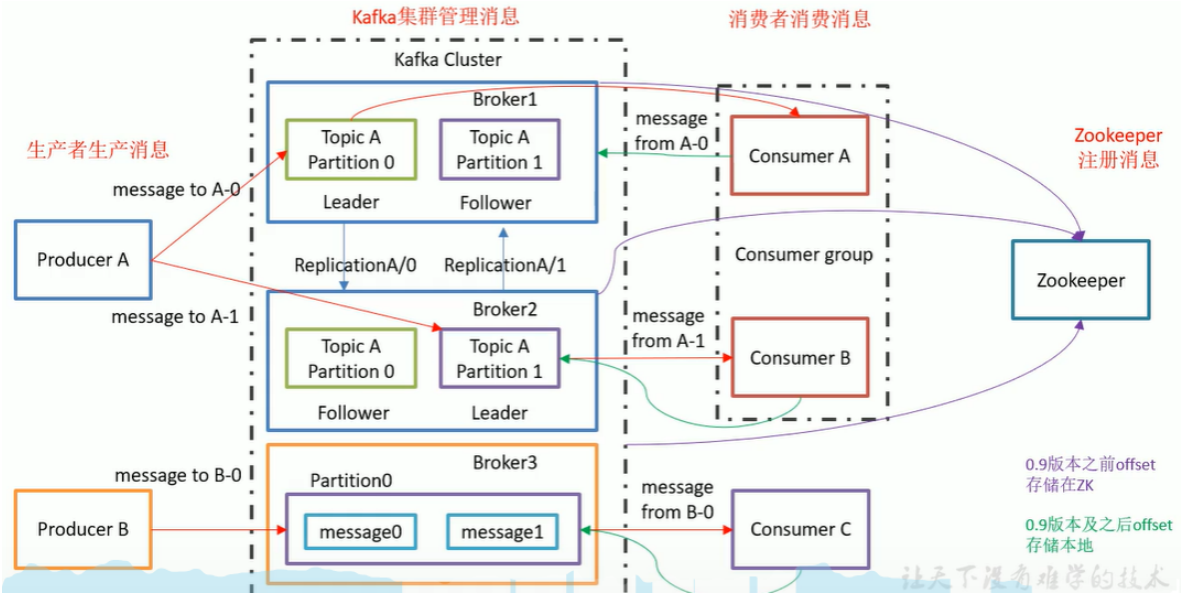
1.1 定义

分布式基于发布/订阅模式的消息队列，用于大数据实时处理领域。

功能：解耦、异步、削峰。

1.2 Kafka基础架构

消息队列两种模式：点到点、发布订阅模式（推和拉两种模式）。Kafka基于Pull模式。



zk作用：1）Kafka集群工作，支撑Leader选举；2）0.9版本前消费者的offset（0.9后存储在Kafka磁盘中），降低对ZK的频繁交互；

Leader和Follower：面向Topic，而不是面向Broker。

broker：一个服务器是一个broker；一个集群由多个broker组成；一个broker可容纳多个topic。

Consumer Group：消费者组内每个消费者负责消费不同分区的数据。一个分区只能一个组内消费者消费。消费者组间互不影响。

Partition: 为实现扩展性，一个topic可分为多个Partition。每个partition是一个有序的队列。

Replica: 副本

二、安装部署

2.1 集群规划

多个节点，每个节点部署zk和kafka

hadoop102	hadoop103	hadoop104
zk	zk	zk
kafka	kafka	kafka

2.2 配置文件

Kafka集群中所有broker修改server.properties，同时修改连接的zk信息

```
broker.id=0  #全局唯一
delete.topic.enable=true
log.dirs=xxx  # topic的消息数据
log.retention.hours=168  #消息数据保存时间，默认7天
log.segment.bytes=1073141824 # 默认1G
# 配置zk集群信息，集群信息，需写所有zk节点2181信息；从而ZK知道哪些kafka是同一个集群
zookeeper.connect=localhost:2181
zookeeper.connect.timeout.ms=6000
```

2.3 启停过程

1、启动zk集群

每个zk节点，单独的zoo.cfg修改（如果不使用kafka自带的zk）

2、每个broker启停Kafka

```
常用命令kafka/bin目录
kafka-server-start.sh
kafka-server-stop.sh
# 如下测试使用
kafka-topics.sh
kafka-console-consumer.sh
kafka-console-producer.sh
```

启动 kafka-server-start.sh -daemon config/server.properties

停止 kafka-server-stop.sh

2.4 topic命令操作

> --zookeeper 必须指定，指定集群信息

1. 查看当前服务所有topic

```
kafka-topics.sh --zookeeper hadoop102:2181 --list
```

2. 创建topic（指定分区和副本数）

```
# 指定副本，分区
kafka-topics.sh --zookeeper hadoop102:2181 --create --replication-facotr 2 --partitions 2 -- topic first
```

创建后会在不同机器log.dir目录下生成分区文件，遍历所有节点在会发现副本数。共计会发现两次first-0,first-1

```
[atguigu@hadoop102 logs]$ ll
总用量 148
-rw-rw-r--. 1 atguigu atguigu     0 7月  12 15:37 cleaner-offset-checkpoint
-rw-rw-r--. 1 atguigu atguigu 22950 7月  12 15:48 controller.log
drwxrwxr-x. 2 atguigu atguigu  4096 7月  12 15:48 first-1
-rw-rw-r--. 1 atguigu atguigu     0 7月  12 15:37 kafka-authorizer.log
```

如果同时创建多个topic，kafka日志存储形式类似如下：

```
[atguigu@hadoop102 kafka]$ ll logs/
总用量 256
-rw-rw-r--. 1 atguigu atguigu     4 7月  12 15:52 cleaner-offset-checkpoint
-rw-rw-r--. 1 atguigu atguigu 42777 7月  12 15:58 controller.log
drwxrwxr-x. 2 atguigu atguigu  4096 7月  12 15:54 first-2
-rw-rw-r--. 1 atguigu atguigu     0 7月  12 15:37 kafka-authorizer.log
-rw-rw-r--. 1 atguigu atguigu     0 7月  12 15:37 kafka-request.log
-rw-rw-r--. 1 atguigu atguigu  7662 7月  12 15:52 kafkaServer-gc.log.0.current
-rw-rw-r--. 1 atguigu atguigu 29164 7月  12 15:57 kafkaServer.out
-rw-rw-r--. 1 atguigu atguigu  1581 7月  12 15:52 log-cleaner.log
-rw-rw-r--. 1 atguigu atguigu     4 7月  12 15:59 log-start-offset-checkpoint
-rw-rw-r--. 1 atguigu atguigu    54 7月  12 15:37 meta.properties
-rw-rw-r--. 1 atguigu atguigu    58 7月  12 15:59 recovery-point-offset-checkpoint
-rw-rw-r--. 1 atguigu atguigu    58 7月  12 15:59 replication-offset-checkpoint
drwxrwxr-x. 2 atguigu atguigu  4096 7月  12 15:57 second-0
drwxrwxr-x. 2 atguigu atguigu  4096 7月  12 15:57 second-1
drwxrwxr-x. 2 atguigu atguigu  4096 7月  12 15:57 second-2
drwxrwxr-x. 2 atguigu atguigu  4096 7月  12 15:57 second-3
-rw-rw-r--. 1 atguigu atguigu 75745 7月  12 15:57 server.log
-rw-rw-r--. 1 atguigu atguigu 37893 7月  12 15:57 state-change.log
```

3. 删除topic

```
# 需要server.properties中delete.topics.enable=true,否则只是标记删除
kafka-topics.sh --zookeeper hadoop102:2181 --delete --topic first
```

4. 查看某个topic详情

```
kafka-topics.sh --zookeeper hadoop102:2181 --describe --topic first
```

```
[atguigu@hadoop102 kafka]$ bin/kafka-topics.sh --describe --topic first --zookeeper hadoop10
81
Topic:first     PartitionCount:3        ReplicationFactor:1     Configs:
        Topic: first    Partition: 0    Leader: 1       Replicas: 1     Isr: 1
        Topic: first    Partition: 1    Leader: 2       Replicas: 2     Isr: 2
        Topic: first    Partition: 2    Leader: 0       Replicas: 0     Isr: 0
```

5. 修改分区数

```
kafka-topics.sh --zookeeper hadoopo102:2181 --alter -topic first --partition
6
```

2.5 生产者和消费者(自带shell)

生产者：

```
kafka-console-producer.sh --topic first --zookeeper hadoop1:2181
```

消费者：连Kafka信息

```
kafka-console-consumer.sh --topic first --bootstrap-server hadoop1:9092
```

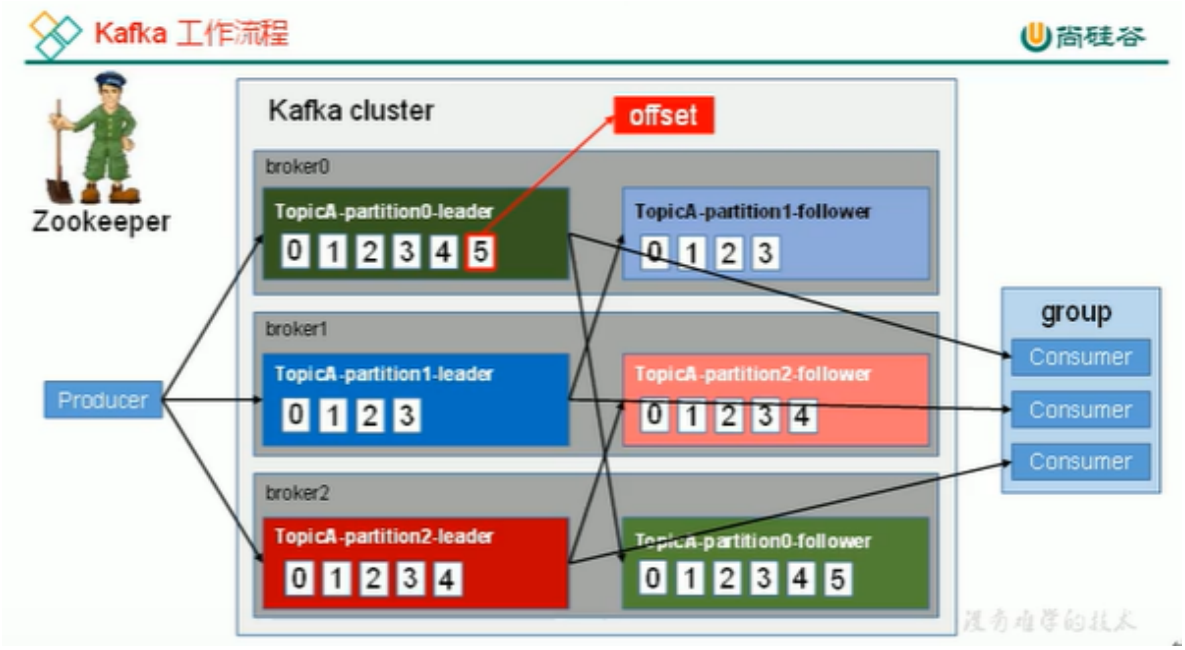
如果消费者加入晚，需要从头开始消费，增加参数 --from-beginning，最早7天内数据

2.6 offset备份在kafka的topic中

```
u     4 7月   12 15:52 cleaner-offset-checkpoint
u  4096 7月   12 16:17 __consumer_offsets-11
u  4096 7月   12 16:17 __consumer_offsets-14
u  4096 7月   12 16:17 __consumer_offsets-17
u  4096 7月   12 16:17 __consumer_offsets-2
u  4096 7月   12 16:17 __consumer_offsets-20
u  4096 7月   12 16:17 __consumer_offsets-23
u  4096 7月   12 16:17 __consumer_offsets-26
u  4096 7月   12 16:17 __consumer_offsets-29
u  4096 7月   12 16:17 __consumer_offsets-32
u  4096 7月   12 16:17 __consumer_offsets-35
u  4096 7月   12 16:17 __consumer_offsets-38
u  4096 7月   12 16:17 __consumer_offsets-41
u  4096 7月   12 16:17 __consumer_offsets-44
u  4096 7月   12 16:17 __consumer_offsets-47
u  4096 7月   12 16:17 __consumer_offsets-5
u  4096 7月   12 16:17 __consumer_offsets-8
u 38092 7月   12 16:17 controller.log
u 42777 7月   12 15:58 controller.log.2019-07-12-15
u  4096 7月   12 16:14 first-2
```

三、Kafka架构深入

3.1 工作流程和文件存储机制



> 上述所有数字只代表offset，不是实际的内容。共计发送了15条消息。

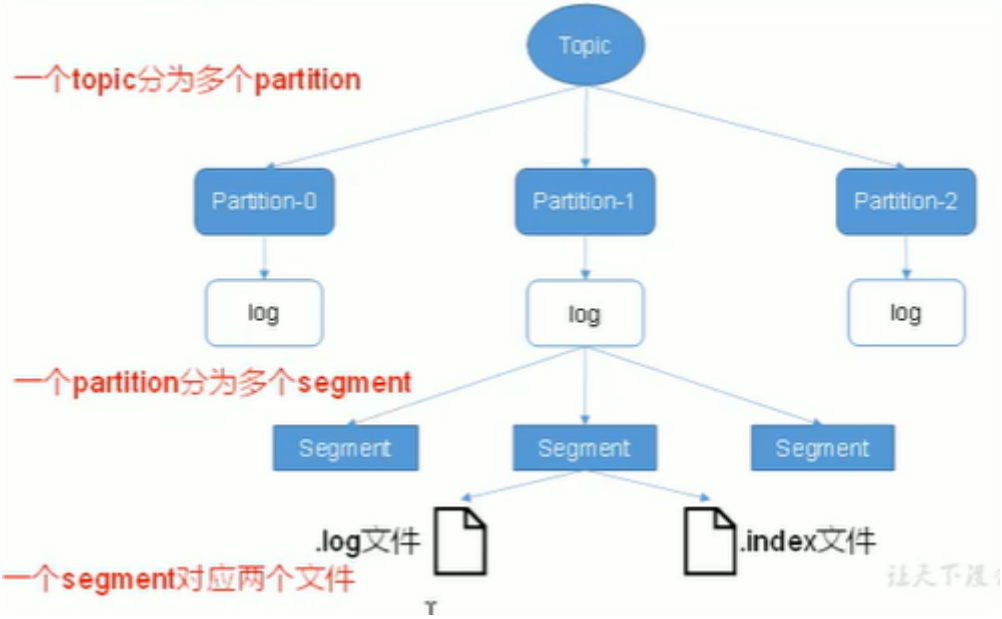
只保证分区内消息有序，不能保证全局消息有序。

实际过程中，每个consumer都会实时记录消费到了哪个offset；出错恢复时可以恢复现场。

3.1.1 partition=n*segment（index+log）

由于生产者消息不断追加到log末尾，为防止log过大而导致数据定位效率问题。所以采用分片+索引机制：

> 1. 将每个partition分为多个segment。
> 2. 每个segment对应文件：index+log，位于同一目录（topic名称+分区号）



1. **每个片段的大小控制：**默认最大1G

```
# The minimum age of a log file to be e
log.retention.hours=168

# A size-based retention policy for log
# segments don't drop below log.retenti
#log.retention.bytes=1073741824

# The maximum size of a log segment fil
log.segment.bytes=1073741824
```

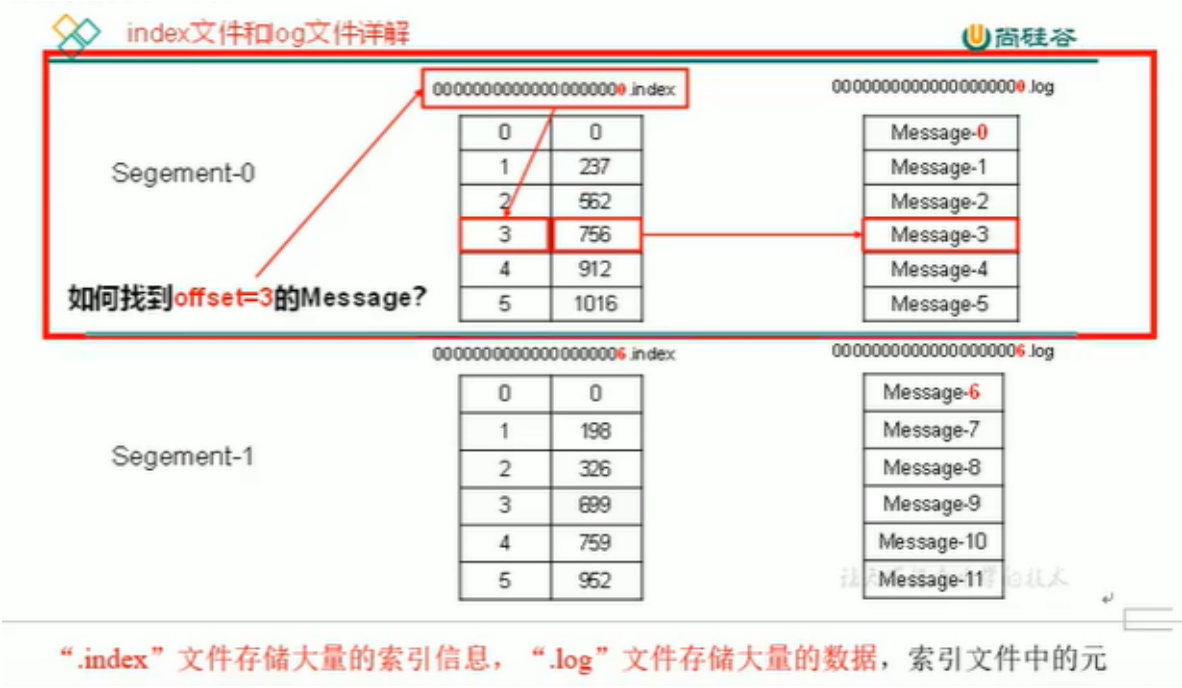
2. index和log所处目录名称：topic名称+分区号

```
[atguigu@hadoop102 data]$ cd first-0/
[atguigu@hadoop102 first-0]$ ll
总用量 16
-rw-rw-r--. 1 atguigu atguigu 10485760 7月  13 09:33 00000000000000000000.index
-rw-rw-r--. 1 atguigu atguigu       73 7月  12 16:32 00000000000000000000.log
-rw-rw-r--. 1 atguigu atguigu 10485756 7月  13 09:33 00000000000000000000.timeindex
-rw-rw-r--. 1 atguigu atguigu       10 7月  12 17:05 00000000000000000001.snapshot
-rw-rw-r--. 1 atguigu atguigu        8 7月  12 16:32 leader-epoch-checkpoint
```

3. index和log以当前segment第1条消息offset命名。

```
00000000000000000000.index
00000000000000000000.log
00000000000000170410.index
00000000000000170410.log
00000000000000239430.index
00000000000000239430.log
```

查找过程示意：1）定位segment；2）根据offset查找消息



3.2 生产者（发送可靠性）

3.2.1 发送partition分区选择

分区原因：

1. 方便集群扩展：每个parition可调整适应所在机器；一个topic可以有多个partition，因此集群可以适应任意大小数据
2. 提高并发：以partition为单位读写

分区原则：

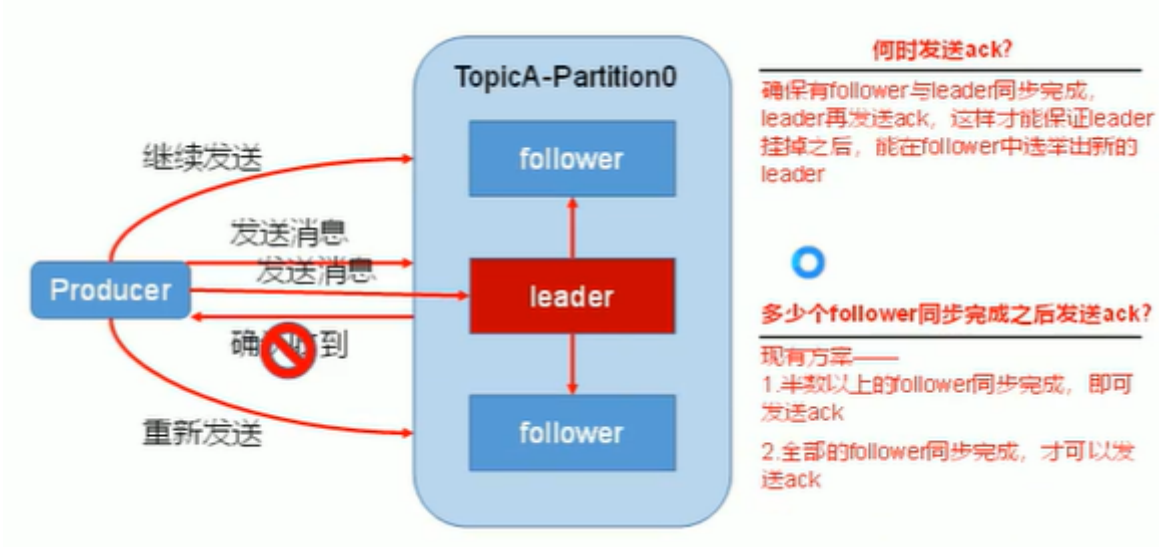
根据源码分析：

1. 直接指定；
2. 根据分区数取余

3.2.2 发送数据可靠性保证

producer成功发送消息的ack机制：

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。



当前方案：半数以上同步完成，就发ack。

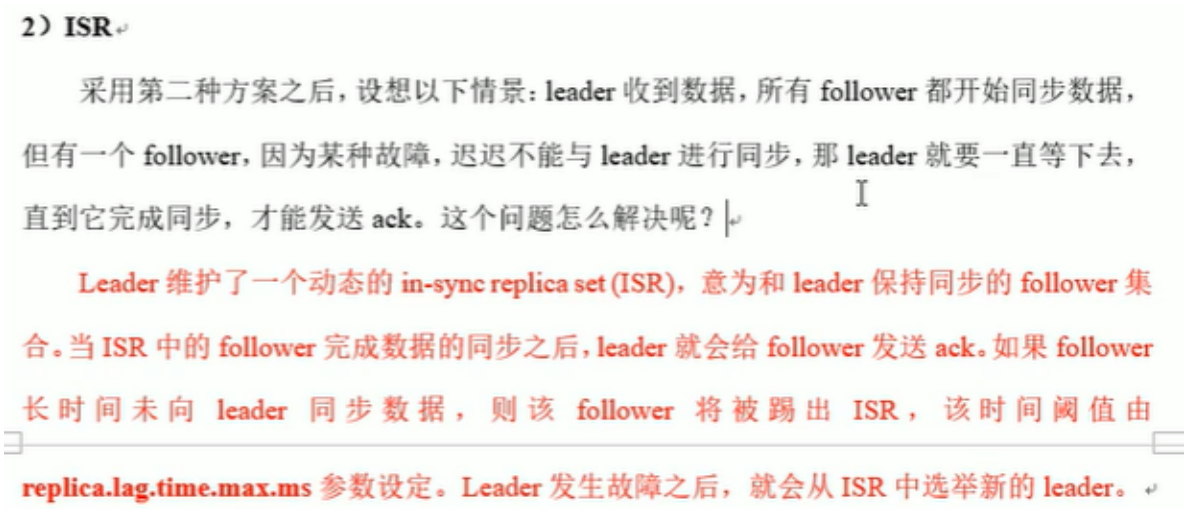
1）副本数据同步策略

方案	优点	缺点
半数以上完成同步，就发送 ack	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

3.2.3 生产者ISR（In sync replica set）

```
[atguigu@hadoop102 kafka]$ bin/kafka-topics.sh --describe --topic first --zookeeper hadoop102:21
81
.Topic:first    PartitionCount:2        ReplicationFactor:2     Configs:
        Topic: first    Partition: 0    Leader: 2       Replicas: 2,0   Isr: 2,0
        Topic: first    Partition: 1    Leader: 0       Replicas: 0,1   Isr: 1,0
[atguigu@hadoop102 kafka]$
```

ISR：记录哪些follower与leader数据一致；后续leader挂掉后，从isr中选取leader；



2）ISR

采用第二种方案之后，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。这个问题怎么解决呢？

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 发生故障之后，就会从 ISR 中选举新的 leader。

P14 15min

https://www.bilibili.com/video/BV1a4411B7V9?p=14

Kafka日志说明