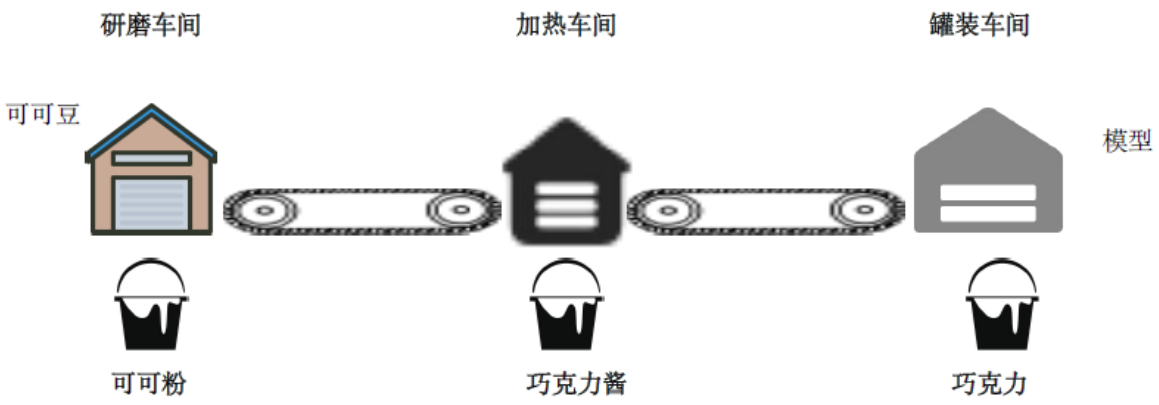
Apache Kafka基础及开发

• Objective(本课目标)

- ☑ Kafka核心概念介绍
- ☑ Kafka安装
- ☑ 了解Kafka的特性
- ☑ 深度剖析Kafka服务端高并发，高性能，高可用的架构设计
- ☑ 深度剖析Kafka提升Consumer的稳定性设计
- ☑ 让大家深入了解Kafka的设计，合理对Kafka进行调优
- ☑ Kafka Producer和Consumer开发
- ☑ 揭秘Kafka Producer高性能架构设计方案

• Kafka核心概念介绍

- 问题：每道工序的生产速度不同，就会导致上下游的工人相互等待！

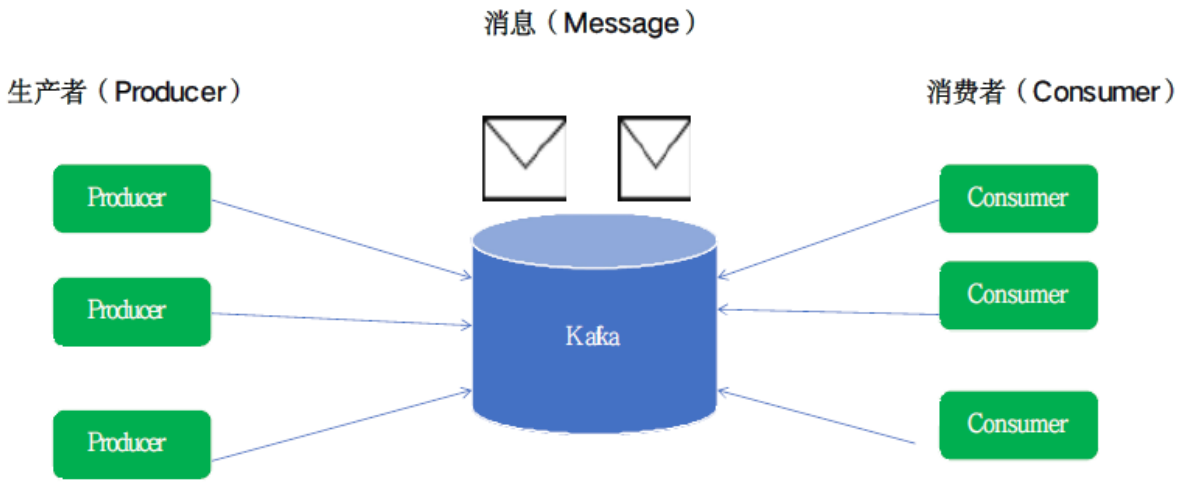


• 为什么会有消息系统

- 仓库的作用就类似于消息系统对于应用程序的作用

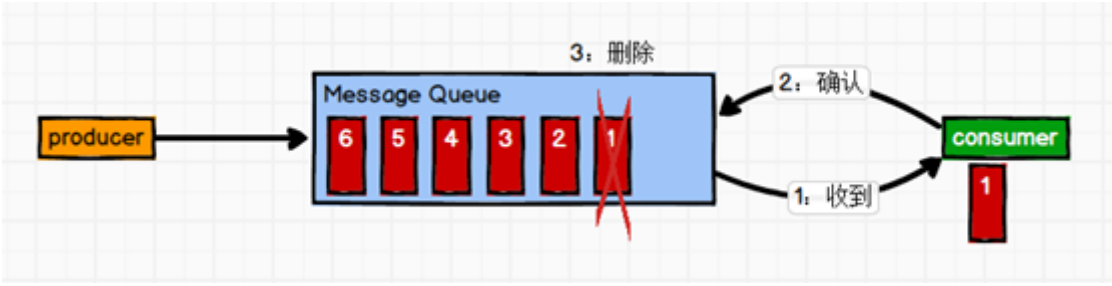
常见的消息系统

- RabbitMQ
- ActiveMQ
- RoketMQ
- Kafka（重点）
- .....
- 为什么学习Kafka?
  - 互联网最火的技术：A（AI）B（BigData）C(Cloud)，B最容易落地
  - 大数据里经常需要应对数据激增，数据复杂度增加，数据变化速率变快等场景
  - Kafka能轻松解决这些问题
- Kafka是一个开源的高吞吐量的分布式发布订阅消息系统

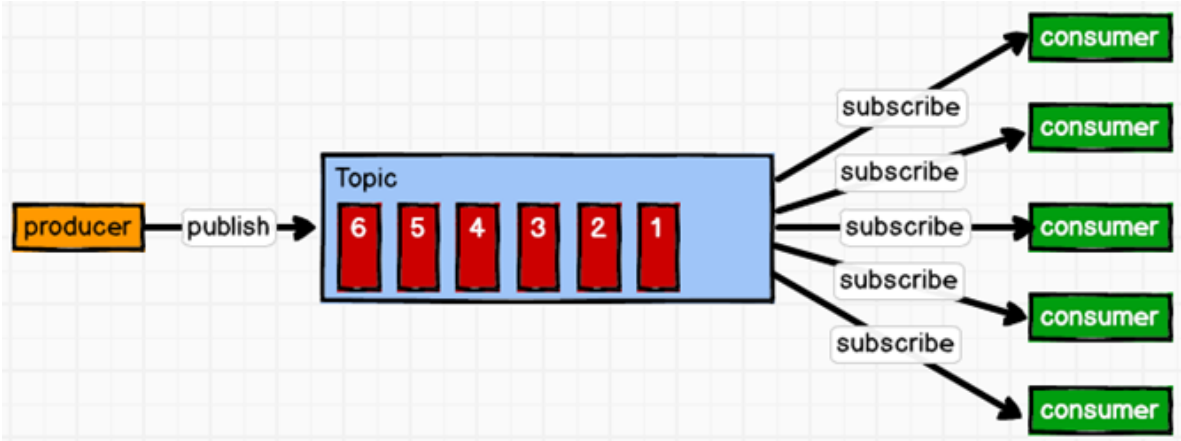


消息队列的两种模式

- **点对点模式**（一对一，消费者主动拉取数据，消息收到后消息清除）
  - 消息生产者生产消息发送到Queue中，然后消费者从Queue中取出并且消费消息。消息被消费以后，queue中不再有存储，所以消息消费者不可能消费到已经被消费的消息。Queue支持存在多个消费者，但是对一个消息而言，只会有一个消费者可以消费。

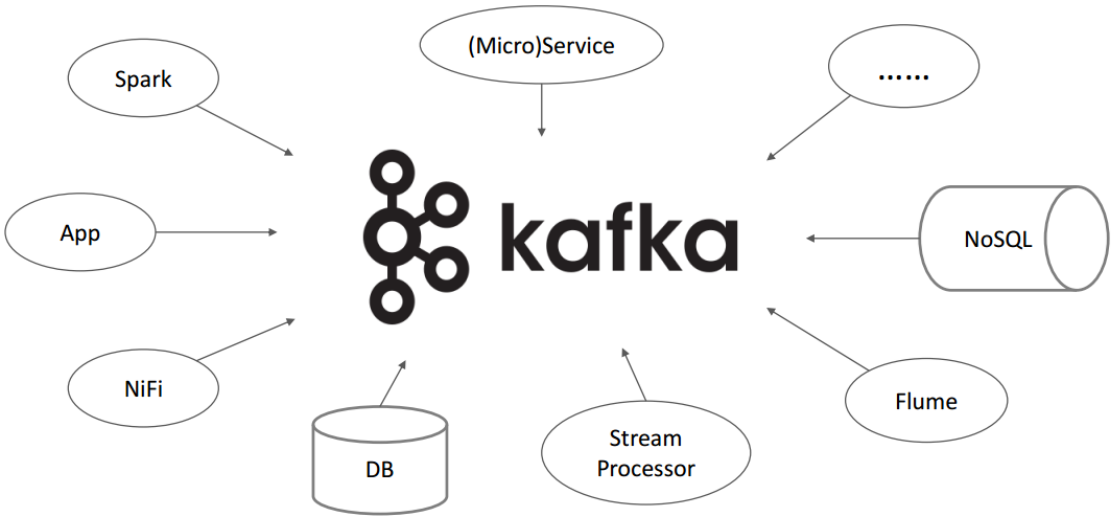


- **发布/订阅模式**（一对多，消费者消费数据之后不会清除消息）
  - 消息生产者（发布）将消息发布到topic中，同时有多个消息消费者（订阅）消费该消息。和点对点方式不同，发布到topic的消息会被所有订阅者消费。

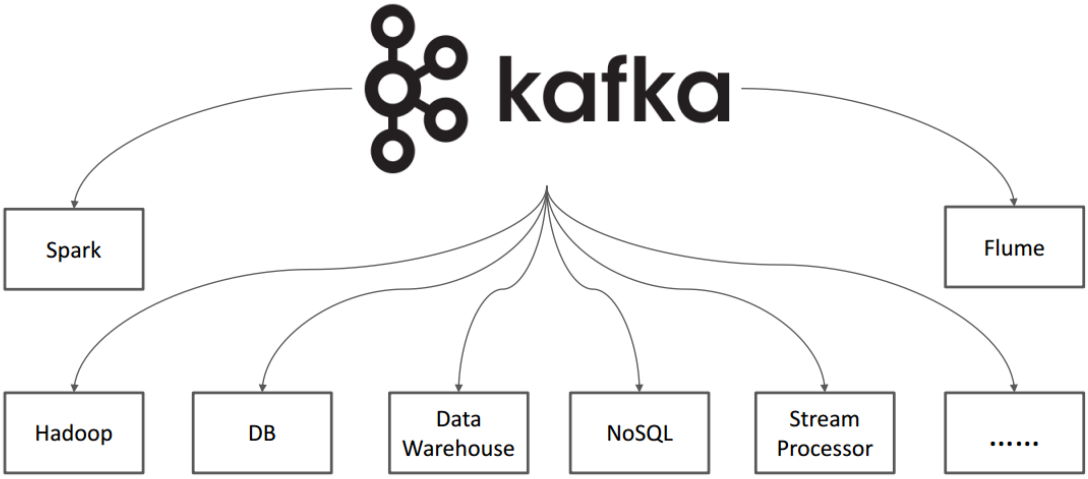


- **Enterprise Messaging with Apache Kafka(使用Kafka的企业消息传递)**

> “
>
> 基于发布与订阅的梢息系统。它一般被称为 “分布式提交日志”或者“分布式流平台”。文件系统或数据库提交日志用来提供所有事务的持久记录， 通过重放这些日志可以重建系统的状态。同样地，Kafka的数据是按照一定顺序持久化保存的，可以按需读取 。 此外，Kafka的数据分布在整个系统里，具备数据故障保护和性能伸缩能力。



- **Then Kafka…**



Apache Kafka

- Apache Kafka is a high-throughput, distributed publish-subscribe messaging system that is designed to be fast, scalable and durable: (Apache Kafka是一个高吞吐量的分布式发布-订阅消息传递系统，它被设计为快速、可伸缩和持久的)
  - Fast:
    - A single Kafka broker(集群的节点) can handle hundreds of megabytes of reads and writes per second from thousands of clients (单个broker每秒可以处理上百兆的数据的读写，可以同时处理上千个客户端的请求)
  - Scalable:(可扩展的)
    - A Kafka cluster can be elastically and transparently expanded without downtime(Kafka集群可以弹性地、透明地扩展，而不需要停机)
  - Durable:(持久)
    - Messages are persisted on disk and replicated within the cluster; (消息保存在磁盘上并在集群中复制)
  - Real-time(实时)
    - Kafka is used for building real-time data pipelines and streaming apps. (Kafka用于构建实时数据管道和流数据处理应用)
- Kafka is written in Scala (Kafka是用Scala写的)

Kafka 安装

- 单机版

```
#第一步：
使用课程提供的 kafka_2.11-0.11.0.2.tgz

#第二步：解压
```

```
tar -zxvf kafka_2.11-0.11.0.2.tgz -C ../install/

#第三步：修改配置
修改kafka_2.11-0.11.0.2/config/server.properties
broker.id=1 # 唯一ID同一集群下broker.id不能重复
listeners=PLAINTEXT://hadoop4:9092    # 监听地址
log.dirs=/root/install/kafka_2.11-0.11.0.2/data       # 数据目录
log.retention.hours=168    # kafka数据保留时间单位为hour 默认 168小时即 7天
zookeeper.connect=hadoop4:2181  #(zookeeper连接ip及port,多个以逗号分隔)

#第四步：配置环境变量
KAFKA_HOME=/root/install/kafka_2.11-0.11.0.2
PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$SCALA_HOME:$SPARK_H
OME:$FLINK_HOME:$HIVE_HOME/bin:$ZOOKEEPER_HOME/bin:$SPARK_HOME/bin:$KAFKA_HOME/bi
n

#第五步：启动服务，先启动zookeeper
./bin/kafka-server-start.sh config/server.propertiess
./bin/kafka-server-stop.sh
```

- 集群版

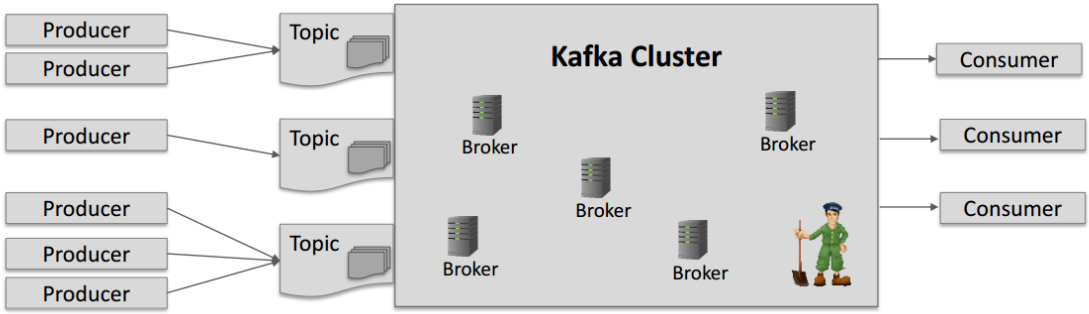
```
前提条件：需要安装zookeeper集群
集群规划：三台节点的集群
    hadoop1：broker
    hadoop2：broker
    hadoop3：broker

第一步：解压
tar -zxvf kafka_2.11-0.11.0.2.tgz -C ../install/

第二步：修改kafka_2.11-0.11.0.2/config/server.properties
#broker的全局唯一编号，不能重复
broker.id=1
#删除topic功能使能
delete.topic.enable=true
#处理网络请求的线程数量 num.network.threads=3
#用来处理磁盘IO的现成数量 num.io.threads=8
#发送套接字的缓冲区大小 socket.send.buffer.bytes=102400
#接收套接字的缓冲区大小 socket.receive.buffer.bytes=102400
#请求套接字的缓冲区大小 socket.request.max.bytes=104857600
#kafka数据的存放地址，多个地址的话用逗号分割/data/kafka-logs-1，/data/kafka-logs-2
log.dirs=/root/install/kafka_2.11-0.11.0.2/data
#新建Topic的默认Partition数量
num.partitions=1
#用来恢复和清理data下数据的线程数量 num.recovery.threads.per.data.dir=1
#segment文件保留的最长时间，超时将被删除 log.retention.hours=168
#配置连接Zookeeper集群地址
zookeeper.connect=hadoop1:2181,hadoop2:2181,hadoop3:2181
```

```
第三步：分发安装包
scp -r kafka_2.11-0.11.0.2 root@hadoop2:/root/install
scp -r kafka_2.11-0.11.0.2 root@hadoop3:/root/install

第四步：修改broker.id
hadoop2 -> broker.id=2
listeners=PLAINTEXT://hadoop2:9092

hadoop3 -> broker.id=3
listeners=PLAINTEXT://hadoop3:9092

第五步：配置环境变量
KAFKA_HOME=/root/install/kafka_2.11-0.11.0.2
PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME:$SCALA_HOME:$SPARK_HOME:$ZOOKEEPER_HOME/bi
n:$KAFKA_HOME/bin

第六步：启动集群
1.先启动zookeeper集群
2.
```

- **Kafka命令操作**

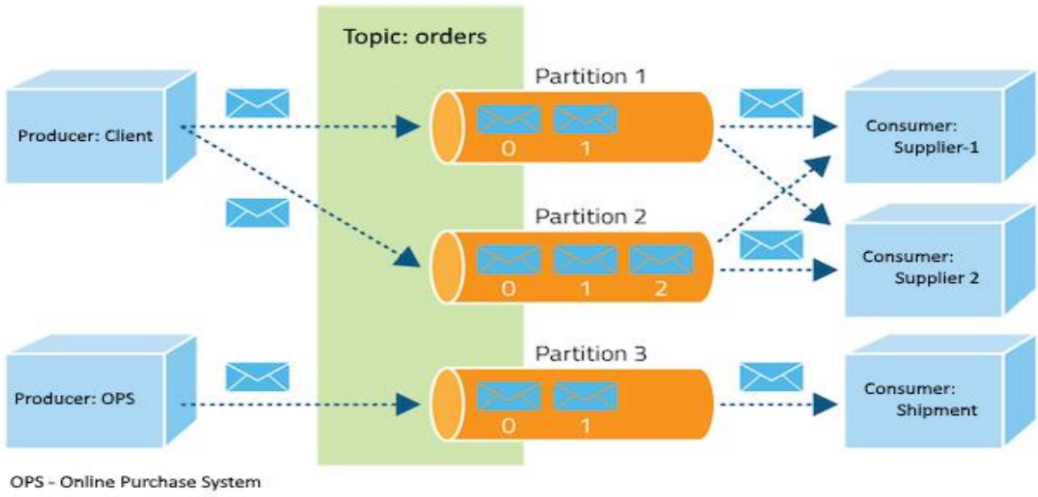
```
#查看 topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --list
kafka-topics.sh --zookeeper hadoop4:2181 --list

#创建 Topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --create --topic
demo --partitions 1 --replication-factor 1

#删除Topic
kafka-topics.sh --zookeeper sandbox-hdp.hortonworks.com:2181 --describe --topic
demo

#发送消息
bin/kafka-console-producer.sh --broker-list hadoop1:9092 --topic demo

#消费消息
bin/kafka-console-consumer.sh --bootstrap-server hadoop1:9092 --from-beginning --
topic demo

#查看某个Topic的详情
bin/kafka-topics.sh --zookeeper hadoop1:2181 --describe --topic demo

#修改分区数
bin/kafka-topics.sh --zookeeper hadoop1:2181 --alter --topic demo --partitions 3

#消息监控
kafka-console-consumer.sh --bootstrap-server Hadoop1:6667 --topic users --from-
beginning
```

Kafka Architecture(架构)



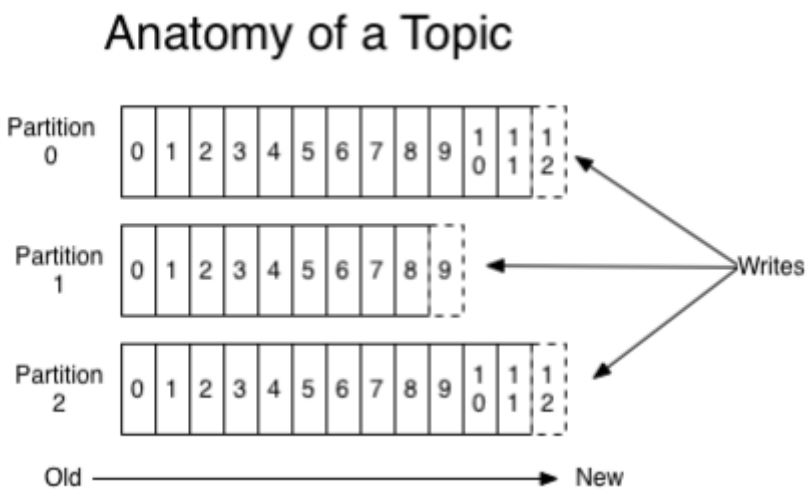
- **Producer：** 消息生产者，就是向Topic发消息的客户端；
- **Consumer：** 消息消费者，向Topic取消息的客户端；
- **Consumer Group （CG）：** 消费者组，由多个consumer组成。**消费者组内每个消费者负责消费不同分区的数据，一个分区只能由一个消费者消费；消费者组之间互不影响。** 所有的消费者都属于某个消费者组，即**消费者组是逻辑上的一个订阅者。**
- **Broker：** 一台kafka服务器就是一个broker。一个集群由多个broker组成。一个broker可以容纳多个topic。
- **Topic：** 可以理解为一个队列，**生产者和消费者面向的都是一个topic**；
- Servers in a Kafka cluster are called Brokers.(Kafka集群中的服务器称为Brokers)

Kafka Topic(主题)

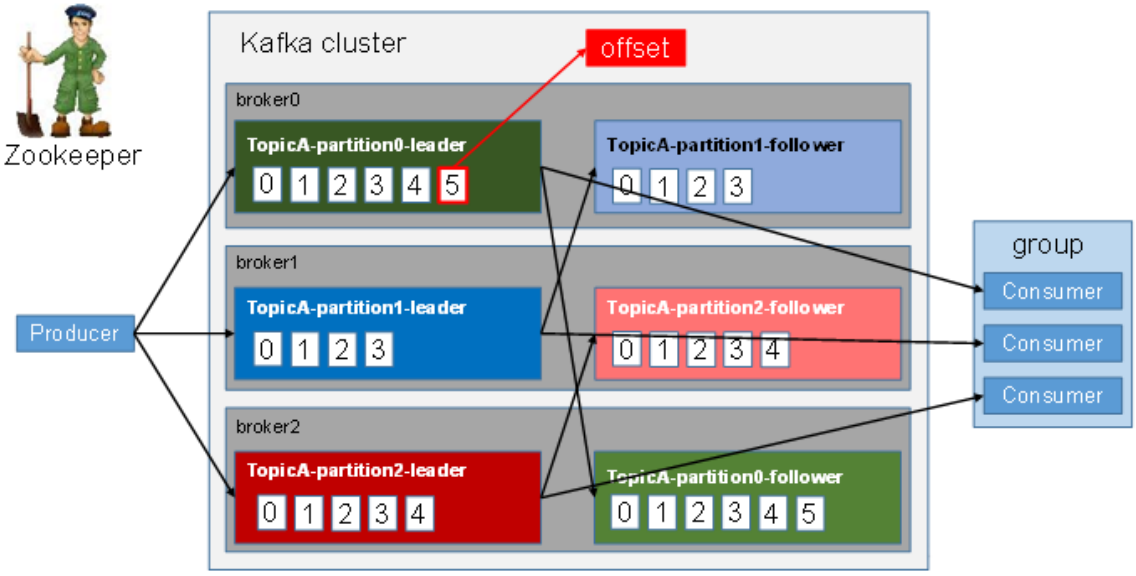


- Topics:
  - A topic is a category or feed name to which messages are published(topic是一类消息的统一名称)
  - 一个Topic可以横跨多个服务器
- Partitions:(分区)
  - Topics are partitioned and replicated across multiple nodes/brokers;(分区和分区副本保存在服务器中不同的节点)

    - 一个非常大的topic可以分布到多个broker（即服务器）上，一个topic可以分为多个partition，每个partition是一个有序的队列
- **Replica：**副本，为保证集群中的某个节点发生故障时，该节点上的partition数据不丢失，且kafka仍然能够继续工作，kafka提供了副本机制，一个topic的每个分区都有若干个副本，一个**leader**和若干个**follower**。
- **leader：**每个分区多个副本的“主”，生产者发送数据的对象，以及消费者消费数据的对象都是leader。
- **follower：**每个分区多个副本中的“从”，实时从leader中同步数据，保持和leader数据的同步。leader发生故障时，某个follower会成为新的follower。
- Logs(message):(日志)
    - Each partition is treated as a log – an ordered set of messages;(每个分区都被视为日志–消息的有序集合)
    - 先进先出的顺序读取
- Retention Period(保留期限)
    - Kafka retains all messages for a configured set amount of time, no matter messages consumed or not;(无论是否消费消息，Kafka都会将所有消息保留一定的配置时间)
- Consumers maintain and track their locations/offsets in each log(消费者在每个日志中维护和跟踪他们的位置/偏移)
- 如何选择分区数量?
    - Topic需要达到多大的吞吐量？例如，是希望每秒钟写入 100KB 还是1GB?
    - 从单个分区读取数据的最大吞吐量是多少？每个分区一般都会有一个消费者，如果你知道消费者将数据写入数据库的速度不会超过每秒50MB，那么你也该知道，从一个分区读取数据的吞吐量不需要超过每秒50MB。
    - 如果你估算出Topic的吞吐量和悄费者吞吐量，可以用Topic吞吐量除以消费者吞吐量算出分区的个数。也就是说，如果每秒钟要从Topic上写入和读取 1GB 的数据，并且每个消费者每秒钟可以处理 50MB 的数据，那么至少需要 20 个分区。这样就可以让20个消费者同时读取这些分区，从而达到每秒钟1GB的吞吐量。
    - 如果不知道以上信息，最好把分区大小限制再25GB以内可以得到比较理想得效果。
- Kafka通常根据时间来决定数据可以被保留多久。默认使用 log.etentlon.hous参数来配置时间，默认值为168小时，也就是一周。

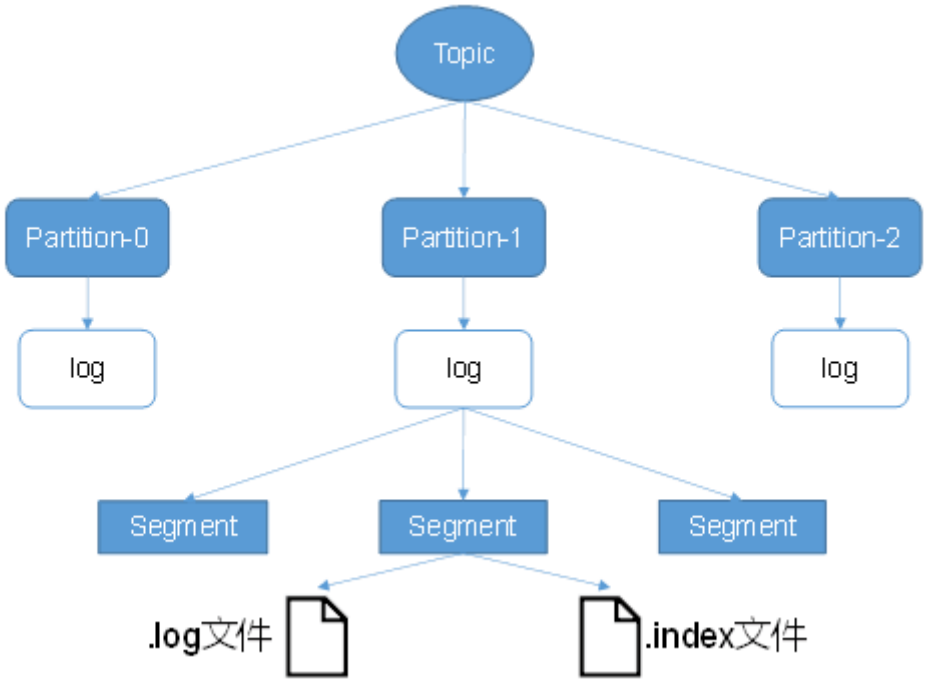


• Kafka架构深入



- Kafka中消息是以**topic**进行分类的，生产者生产消息，消费者消费消息，都是面向topic的。
- topic是逻辑上的概念，而partition是物理上的概念，每个partition对应于一个log文件，该log文件中存储的就是producer生产的数据。Producer生产的数据会被不断追加到该log文件末端，且每条数据都有自己的offset。消费者组中的每个消费者，都会实时记录自己消费到了哪个offset，以便出错恢复时，从上次的位置继续消费。

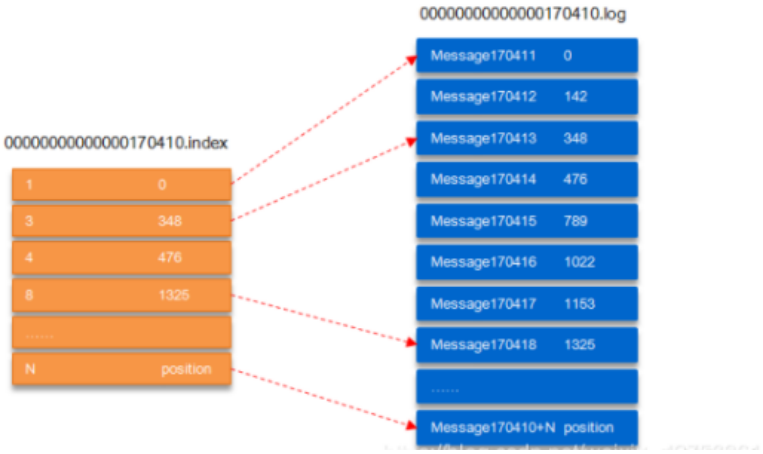
- 文件存储机制



> 由于生产者生产的消息会不断追加到log文件末尾，为防止log文件过大导致数据定位效率低下，Kafka采取了**分片**和**索引**机制，将每个partition分为多个segment。每个segment对应两个文件——“.index”文件和“.log”文件。这些文件位于一个文件夹下，该文件夹的命名规则为：topic名称+分区序号。例如，demo这个topic有三个分区，则其对应的文件夹为demo-0,demo-1,demo-2
>
> 00000000000000000000.index
>
> 00000000000000000000.log
>
> 00000000000000123156.index
>
> 00000000000000123156.log
>
> 00000000000000315215.index
>
> 00000000000000315215.log

- index和log文件以当前segment的第一条消息的offset命名。下图为index文件和log文件的结构示意图。
- “.index”文件存储大量的索引信息，“.log”文件存储大量的数据，索引文件中的元数据指向对应数据文件中message的物理偏移地址。

假设这个消息中有N条消息，第3条消息在index文件中对应的是348，也就是说在log文件中，第3条消息的偏移量为348。

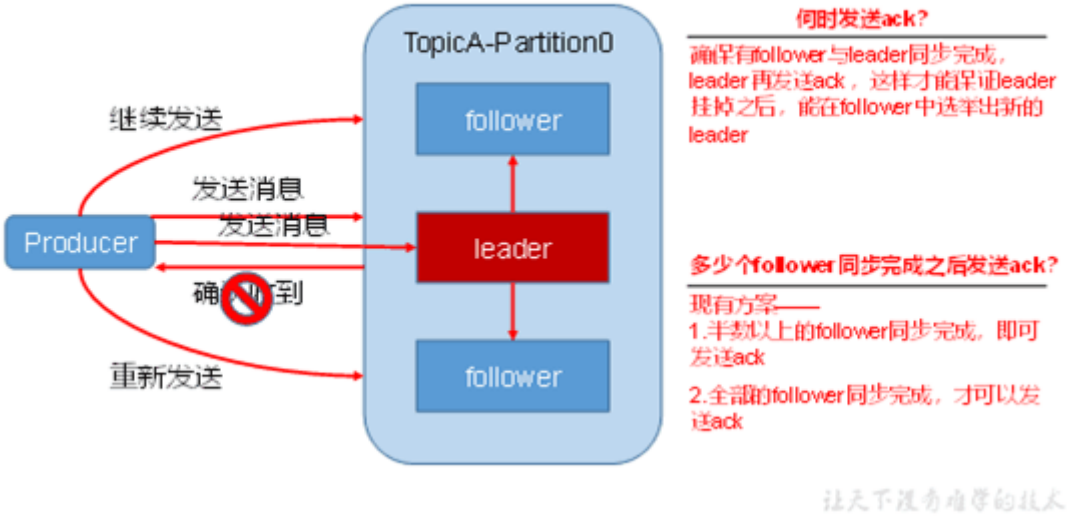


• Kafka Roles - Producer(生产者)

- Producers publish data to the topics of their choice;(生产者将数据发布到他们选择的Topic)
  - A producer is responsible for choosing which message to assign to which partition within the topic; （producer负责选择将哪个消息分配给topic中的哪个分区）
  - **可以提高并发**，因为可以以Partition为单位读写了。
- **ProducerRecord**对象：需要将producer发送的数据封装成一个**ProducerRecord**对象
  - 指明Partition的情况下，直接将指明的值作为partiton值
  - 没有指明partition值但有key的情况下，将Key的hash值与topic的partition数进行取余得到partition值
  - 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 总数取余得到 partition 值，也就是常说的 round-robin 算法。

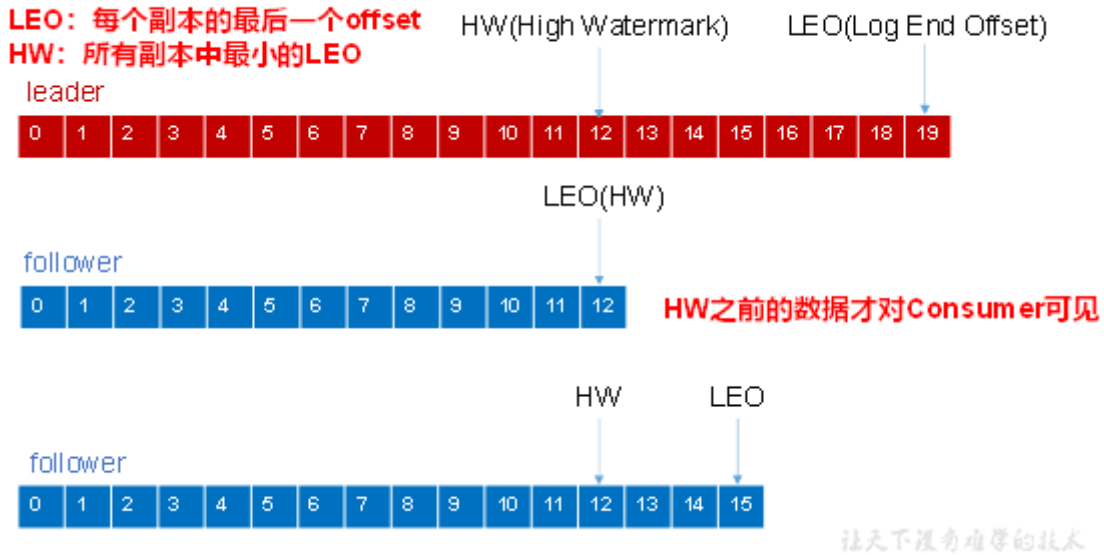
- 数据可靠性保证

- 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**
  - 半数以上完成同步，就发送ack
    - 优点：延迟低
    - 缺点：选举新的leader时，容忍n台节点的故障，需要2n+1个副本（半数）
  - 全部完成同步，才发送ack
    - 优点：选举新的leader时，容忍n台节点的故障，需要n+1个副本
    - 缺点：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量数据的冗余。
    - 虽然第二种方案的网络延迟会比较高，但网络延迟对Kafka的影响较小。
- **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。
- **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**：

    - 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故障时有可能**丢失数据**；
    - 1：producer等待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。但是如果在follower同步完成后，broker发送ack之前，leader发生故障，那么会造成**数据重复**。
- **故障处理细节**



- **follower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。
- **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，其余的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者不重复。**

Exactly Once语义

- 对于某些比较重要的消息，我们需要保证exactly once语义，即保证每条消息被发送且仅被发送一次。
- 在0.11版本之后，Kafka引入了幂等性机制（idempotent），配合acks = -1时的at least once语义，实现了producer到broker的exactly once语义。
    - **idempotent + at least once = exactly once**
- 使用时，只需将enable.idempotence属性设置为true，kafka自动将acks属性设为-1。

Kafka消费者

消费方式

- consumer采用pull（拉）模式从broker中读取数据。
- push（推）模式很难适应消费速率不同的消费者，因为消息发送速率是由broker决定的。它的目标是尽可能以最快速度传递消息，但是这样很容易造成consumer来不及处理消息，典型的表现就是拒绝服务以及网络拥塞。而pull模式则可以根据consumer的消费能力以适当的速率消费消息。
- pull模式不足之处是，如果kafka没有数据，消费者可能会陷入循环中，一直返回空数据。针对这一点，Kafka的消费者在消费数据时会传入一个时长参数timeout，如果当前没有数据可供消费，consumer会等待一段时间之后再返回，这段时长即为timeout。

分区分配策略

- 一个consumer group中有多个consumer，一个 topic有多个partition，所以必然会涉及到partition的分配问题，即确定那个partition由哪个consumer来消费。
- Kafka有两种分配策略，一是roundrobin，一是range。
    - https://blog.csdn.net/lzb348110175/article/details/100773487
- offset的维护
    - 由于consumer在消费过程中可能会出现断电宕机等故障，consumer恢复后，需要从故障前的位置的继续消费，所以consumer需要实时记录自己消费到了哪个offset，以便故障恢复后继续消费。
    - Kafka 0.9版本之前，consumer默认将offset保存在Zookeeper中，从0.9版本开始，consumer默认将offset保存在Kafka一个内置的topic中，该topic为__**consumer_offsets**。

Kafka 高效读写数据

- Kafka的producer生产数据，要写入到log文件中，写的过程是一直追加到文件末端，为顺序写。官网有数据表明，同样的磁盘，顺序写能到到600M/s，而随机写只有100k/s。这与磁盘的机械机构有关，顺序写之所以快，是因为其省去了大量磁头寻址的时间。
- 零拷贝：https://zhuanlan.zhihu.com/p/78335525

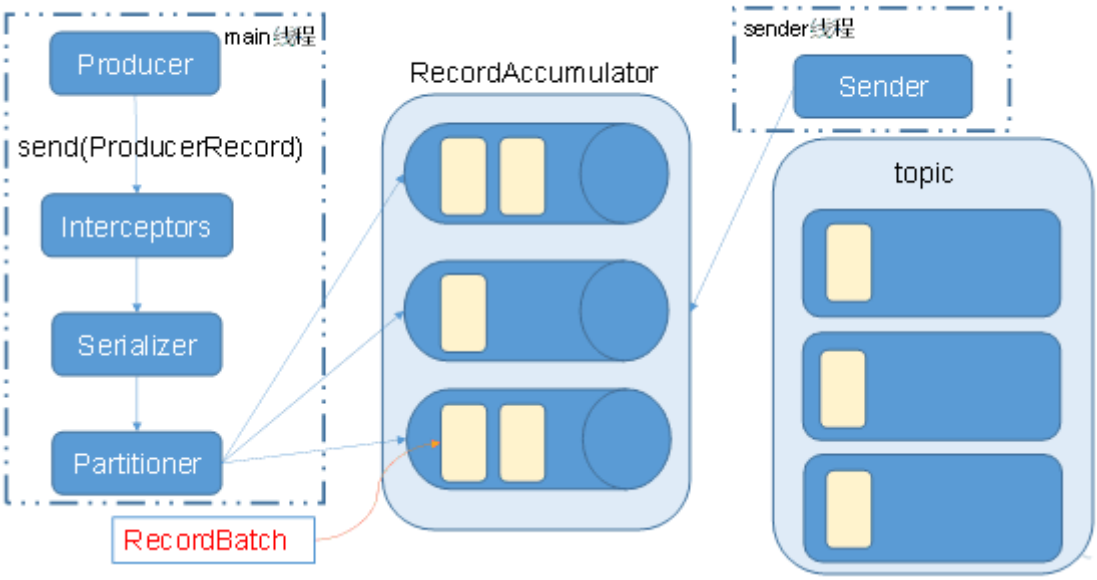
Zookeeper在Kafka中的作用

- Kafka集群中有一个broker会被选举为Controller，负责管理集群broker的上下线，所有topic的分区副本分配和leader选举等工作。
- Controller的管理工作都是依赖于Zookeeper的。
- 以下为partition的leader选举过程：
    - https://www.cnblogs.com/qiu-hua/p/13394377.html

Kafka APIs

消息发送流程

- Kafka的Producer发送消息采用的是**异步发送**的方式。在消息发送的过程中，涉及到了**两个线程——**main线程和Sender线程，**以及**一个线程共享变量——RecordAccumulator**。main线程将消息发送给RecordAccumulator，Sender线程不断从RecordAccumulator中拉取消息发送到Kafka broker。



相关参数：

batch.size：只有数据积累到batch.size之后，sender才会发送数据。

linger.ms：如果数据迟迟未达到batch.size，sender等待linger.time之后就会发送数据。

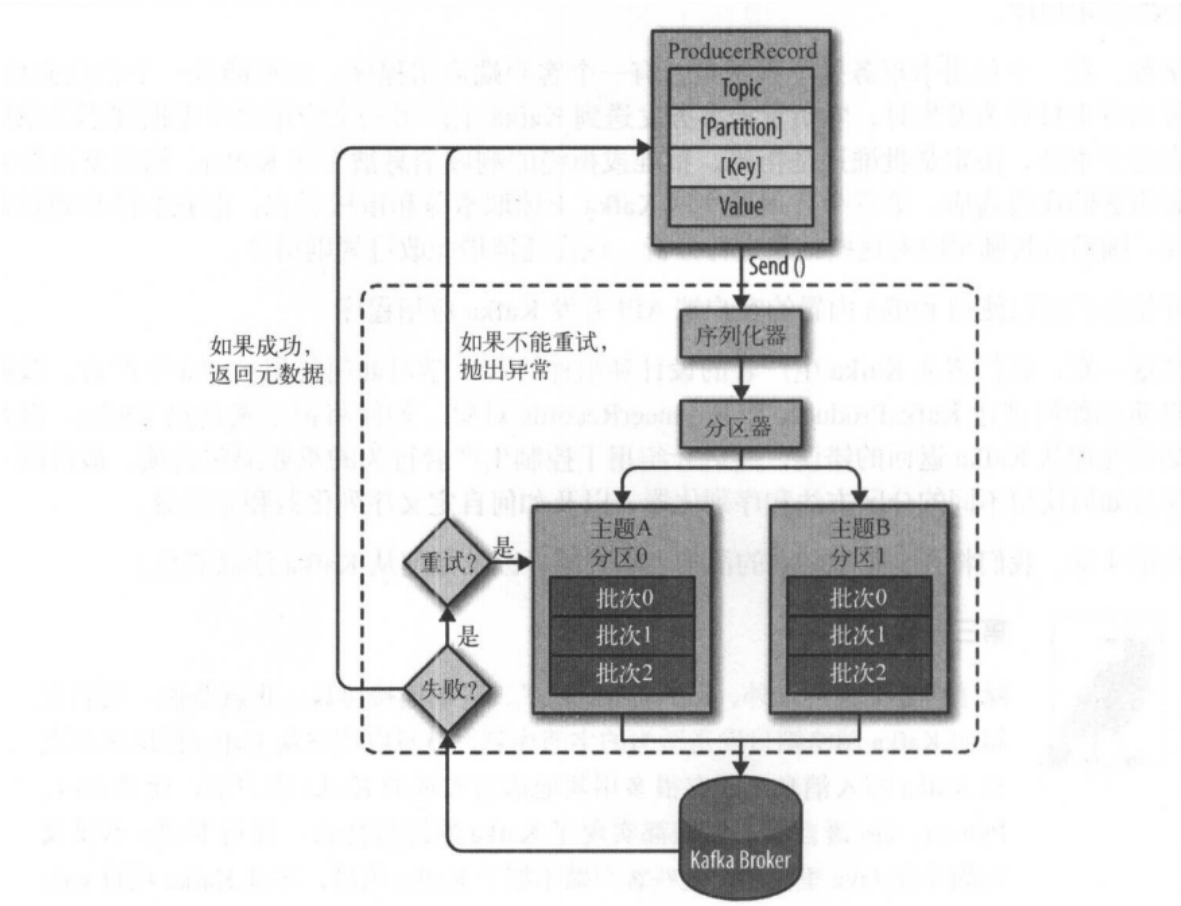
异步发送API

```
案例1 -> job1
KafkaProducer：需要创建一个生产者对象，用来发送数据
ProducerConfig：获取所需的一系列配置参数
ProducerRecord：每条数据都要封装成一个ProducerRecord对象

public static void main(String[] args) throws ExecutionException,
InterruptedException {
        Properties props = new Properties();
        props.put("bootstrap.servers", "hadoop102:9092");//kafka集群，broker-list
        props.put("acks", "all");
        props.put("retries", 1);//重试次数
        props.put("batch.size", 16384);//批次大小
        props.put("linger.ms", 1);//等待时间
        props.put("buffer.memory", 33554432);//RecordAccumulator缓冲区大小
        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 < 1000; i++) {
            producer.send(new ProducerRecord<String, String>("demo",
Integer.toString(i), Integer.toString(i)));
        }
        producer.close();
    }


//回调函数
Producer<String, String> producer = new KafkaProducer<>(props);
        for (int i = 0; i < 100; i++) {
            producer.send(new ProducerRecord<String, String>("first",
Integer.toString(i), Integer.toString(i)), new Callback() {

                //回调函数，该方法会在Producer收到ack时调用，为异步调用
                @Override
                public void onCompletion(RecordMetadata metadata, Exception
exception) {
                    if (exception == null) {
                        System.out.println("success->" + metadata.offset());
                    } else {
                        exception.printStackTrace();
                    }
                }
            });
        }
```

- Async Publishing. (异步发送)
  - acks=0(消息备份的数目，如果是0的话百分之百是异步)
  - acks=1(至少有一个备份，表示写数据完成)
  - acks=3(3个备份都完成，表示写数据完成，安全性最好)
- All nodes can answer metadata requests about:　(所有节点都可以响应以下元数据请求)
  - Which servers are alive (哪些服务器是活动的)
  - Where leaders are for the partitions of a topic (topic中partition的领导者在哪里)

- 消息发送的三种方式
  - fire-and-forget：我们把消息发送给服务器，但并不关心它是否正常到达。大多数情况下，消息会正常到 达，因为 Kafka 是高可用的，而且生产者会自动尝试重发。不过，使用这种方式有时候也会丢失一些消息
  - 同步发送：我们使用send方怯发送消息，它会返回Futue对象，调用 get方法进行等待，就可以知道悄息是否发送成功.
  - 异步发送：我们调用send方怯，并指定一个回调函数，服务器在返回响应时调用该函数.

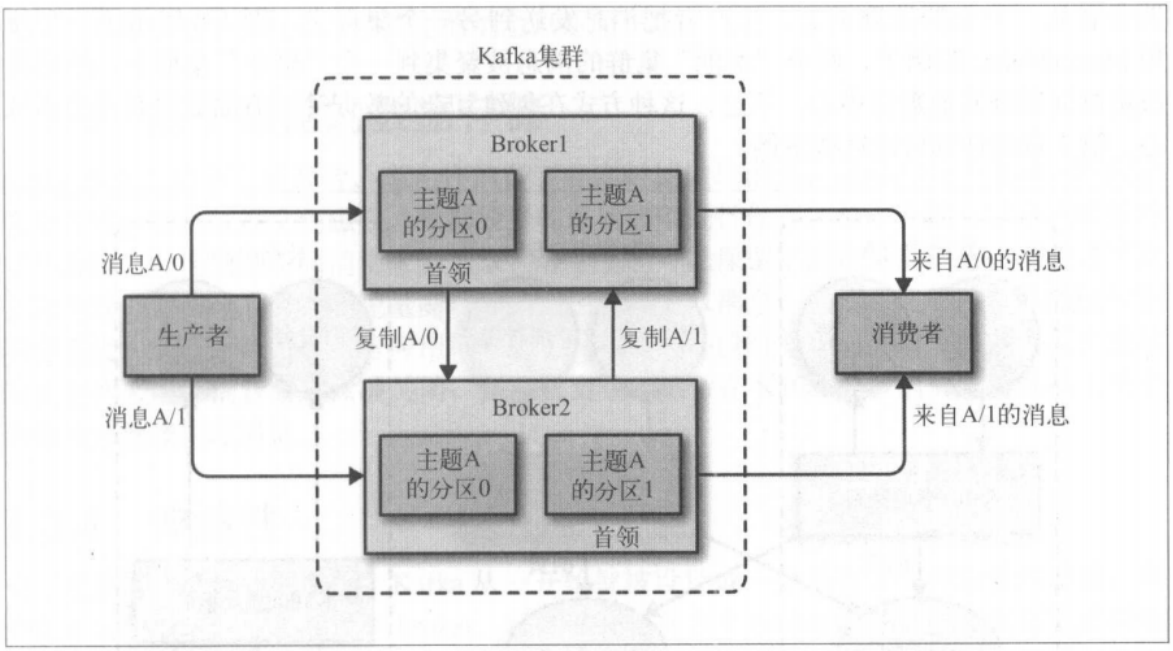



```
//producer的基本操作
//发送消息
./kafka-console-producer.sh --broker-list sandbox-hdp.hortonworks.com:6667 --
topic demo

//接受消息
./kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 -
-topic demo --from-beginning
```

- **Producer – Load Balancing(负载均衡) & ISRs(同步备份)**

- **Kafka Roles - Broker**

- Broker:
  - 单个broker可以轻松处理数千个分区以及每秒百万级的消息量
  - Brokers are servers in a Kafka cluster(brokers是Kafka集群中的服务器)
  - Each partition has one server acting as a leader, and zero or more servers acting as followers / ISRs;(每个分区都有一个服务器充当领导者，零个或多个服务器充当跟随者/ISR)
    - The leader handles all read and write requests. (leader 处理所有的读和写请求)
    - The followers passively replicate the leader (追随者被动地模仿领导者)
- Each server acts as a leader for some partitions and a followers for others, so load is well balanced. (每个服务器充当某些分区的leader 和其他分支的followers ， 因此负载很平衡)
- 保留消息(在一定期限内)是Kafka的一个重要特性。Kafka broker默认的消息保留策略是这样的：要么保留一段时间(比如 7 天)，要么保留到消息达到一定大小的字节数(比如1GB)。当消息数量达到这些上限时，旧消息就会过期并被删除，所以在任何时刻，可用消息的总量都不会超过配置参数所指定的大小.



Kafka Roles - Consumer(消费者)

- Consumers
  - Consumers consume messages through subscriptions; (消费者通过订阅消费消息)
  - Multiple Consumers can read from the same topics (多个consumer可以从相同的topic中读取内容)
    - consumer和topic是多对多得关系
  - Consumers are organized into Consumer Groups; (消费者被组织成消费者组)
  - Kafka offers messages to Consumer Groups, not Consumer (instance) directly; (kafka提供消息给consumer Group，而不是直接给某个consumer)

        - Therefore offset management is at Consumer Group Level （因此消息管理是在ConsumerGroup Level)
            - 实际工作得consumer和pairtion最好是保持一致，一个组里面的consumer可以小于partition
            - 一个consumer组里面不能同时有两个consumer去读同一个partitoin，一个consumer可以读取不同的partition。
    - Messages remain on Kafka, which are not removed after they are consumed (消息保留在Kafka上，消费后不会被删除)
    - https://www.jianshu.com/p/1f9e18e926f6

Consumer - Rebalancing(负载均衡)

- Rebalancing (by group coordinator).(负载均衡)
    - 如果有3个partition只有两个consumer，kafka会自动协调
    - The partitions of a topic are divided among the consumers of a group (topic的partition在组的使用者之间划分)
    - Upon the availability of consumers in a group, the partitions are reassigned so that each consumer receives a proportional share of the partitions(根据组中消费者的可用性，重新分配分区，以便每个消费者按比例分配分区)
        - The # of consumers cannot be more than # of partitions （consumers不要超过partition的数目)
- Messaging Models:(消息传递模型)
    - Queue: a message goes to one of the consumers(队列:消息发送到其中一个consumer)
        - All consumers are in the same Consumer Group;(所有消费者都在同一个消费者组中;)
    - Publish-Subscribe: a message goes to all consumers(发布-订阅:消息发送给所有consumer)
        - All consumers are assigned to different Consumer Groups;(所有消费者被分配到不同的消费组;)

The Role of ZooKeeper

- Kafka uses Apache ZooKeeper as the distributed configuration store.(Kafka使用Apache ZooKeeper作为分布式配置存储)
    - It forms the backbone of Kafka cluster that continuously monitor the health of the brokers / cluster(它形成了Kafka集群的主干，可以持续监控代理/集群的运行状况)
- Initial version of Kafka uses ZooKeeper for storing the offset information for each consumer; Starting from 0.10, Kafka has its own internal topic for the offset storage.(Kafka的初始版本使用ZooKeeper来存储每个消费者的偏移信息;从0.10开始，Kafka有自己内部的偏移存储topic)

Message Ordering

- Ordering is only guaranteed within a partition for a topic　(仅在topic的分区内保证排序)
- To ensure global ordering for a topic:　(确保topic的全局排序)
  - Group messages in a partition by Key (producer) (按键对分区中的消息进行分组)
  - Configure exactly one consumer instance per partition within a consumer group (在使用者组中的每个分区上恰好配置一个使用者实例)
- Note:
  - Data within a Partition will be stored in the order in which it is written, therefore data read from a Partition will be read in the order for that partition (分区内的数据将按写入顺序存储，因此从分区读取的数据将按该分区的顺序读取)

Message Replication(消息备份)

- Durability can be configured with the Producer configuration – request.required.acks（持久性可以通过生产者配置来配置——request.required.acks)
  - 0 – the producer never waits for an ack (生产者从不等待确认)
  - 1 – the producer gets an ack after the leader replica has received the data (在leader副本收到数据后，生成器获得一个ack)
  - -1 / all – the producer gets an ack after all ISRs (in-sync replication) receives the data (生产者在所有ISRs(同步复制)接收到数据后获得一个ack)

Durability(可用性)	Behaviour(行为)	Per Event Latency(延迟)	Required Acks
Highest	ACK all ISRs have received	Highest	-1 / all
Medium	ACK once the leader has received	Medium	1
Lowest	No ACKs required	Lowest	0

- Minimum available ISR can also be configured such that an error is returned if enough replicas are not available to replicate data(还可以配置最小可用ISR，以便在没有足够的副本用于复制数据时返回错误)

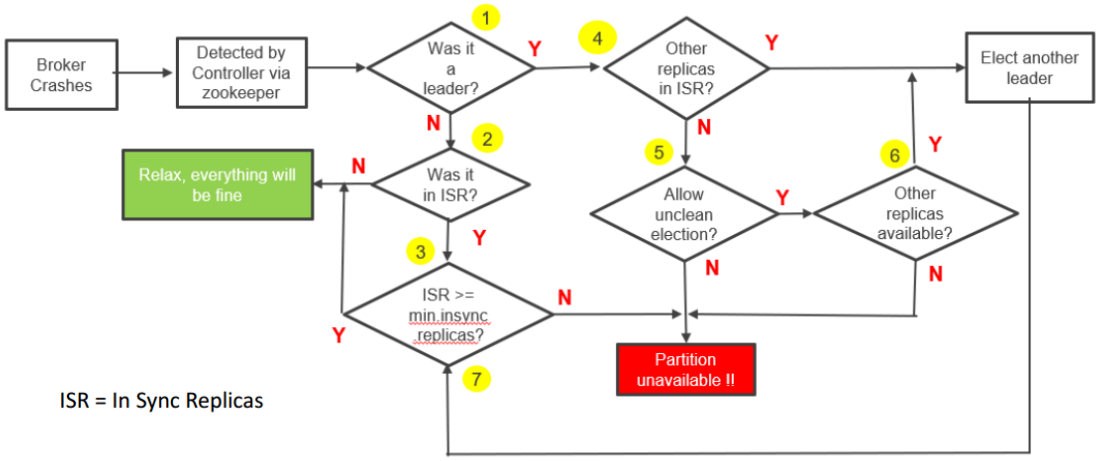
Data Loss at the Producer(Producer数据丢失)

- Kafka Producer API
  - Messages are accumulated in buffer in batches (消息分批地积累在缓冲区中)
  - Messages are batched by partition, retried at batch level (按分区对消息进行批处理，在批处理级别重试)

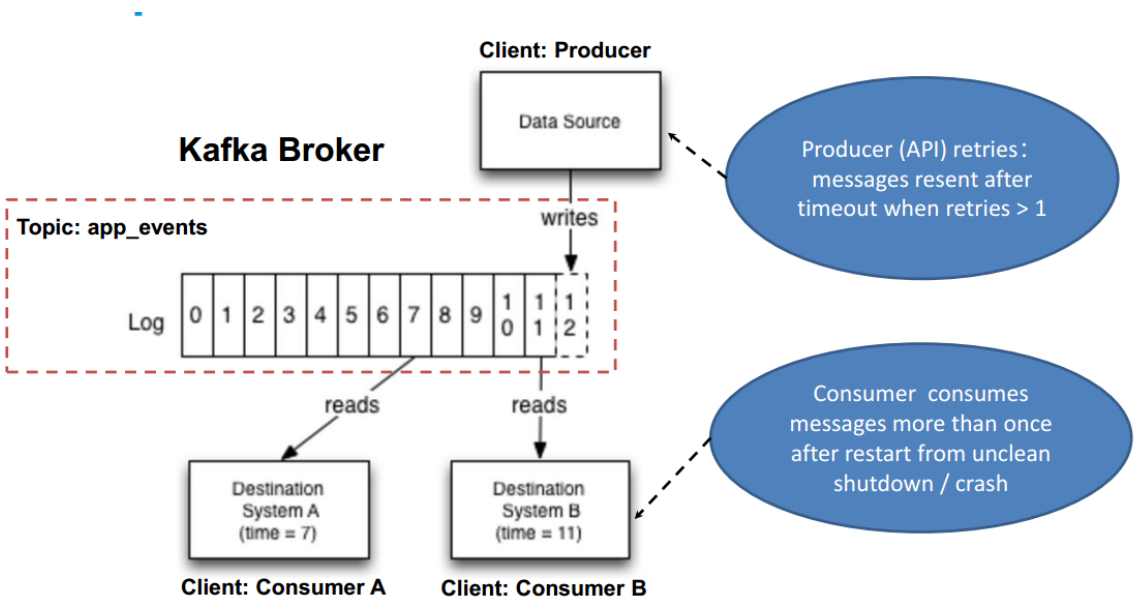
- Expired batches dropped after retries (重试后丢弃的过期批)

- Data Loss at Producer (生产者的数据丢失)
  - Failed to close / flush producer on termination (在终止时close/flush producer失败)
  - Dropped batches due to communication or other errors when acks = 0 or retry exhaustion (当acks = 0或重试耗尽时，由于通信或其他错误而丢失批)
  - Data produced faster than delivery, causing BufferExhausedException (产生的数据比传递的数据快，导致bufferconflicedexception异常)

Data Delivered but Loss in the Cluster(数据提交但群集中丢失)

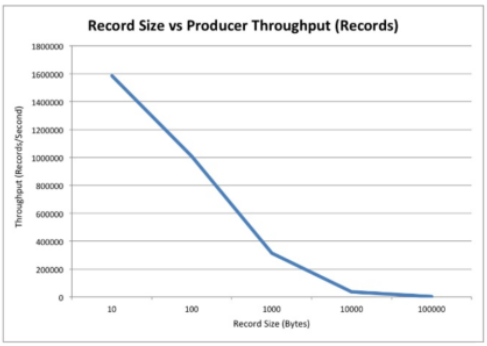


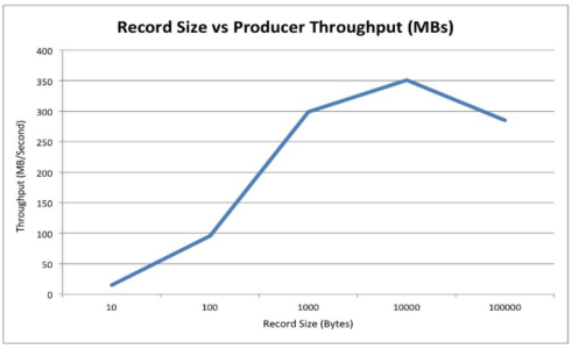
Data Duplication(数据重复)



Producer Performance – Single Thread(生产者性能-单线程)

Type	Records / Sec	MB / Sec	Avg Latency	Max Latency	Median Latency
No Replication	1,100,182	104	42	1070	1
3 x Async	1,056,546	101	42	1157	2
3 x Sync	493,855	47	379	4483	192





From Clouera – http://www.cloudera.com

更可

Summary(总结)

- 了解了Apache Kafka 架构介绍
  - Kafka Components
- 学习了Apache Kafka API 及其分类
  - Producer / Consumer
  - Streaming API
  - Connector API
- 深入学习了Apache Kafka 的工作原理
- 练习了Kafka Producer和Consumer开发