kafka

1. Kafka 是一个分布式的基于发布/订阅模式的可持久化的消息队列（Message Queue），主要应用于大数据实时处理领域。 （AB系统要通信，AB系统不直连，中间经过消息中间间）
2. 消息队列的好处：解耦，可恢复性（部分失效不影响整体），缓冲，削峰，异步
3. kafka基础框架



Kafka

架构

Producer A

Producer B

Topic A

Partition 0

Topic A

Partition 1

Broker1

Leader

Follower

ReplicationA

/0

ReplicationA

/1

Topic A

Partition 0

Topic A

Partition 1

Broker2

Leader

Follower

Consumer A

Consumer B

Kafka Cluster

Broker3

Partition0

message0

message1

Consumer C

Zookeeper

Consumer group

message to A

-

0

message to A

-

1

message to B

-

0

message

from A

-

0

message

from A

-

1

message

from B

-

0

生产者生产消息

Kafka

集群管理消息

消费者消费消息

Zookeeper

注册消息

0.9

版本之前

offset

存储在

ZK

0.9

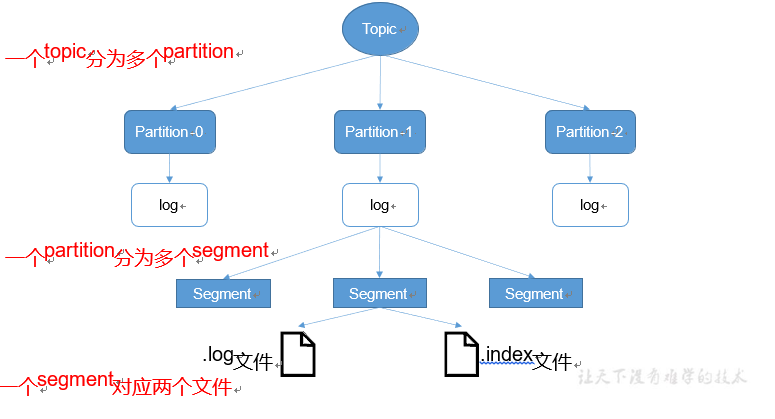
版本及之后

offset

存储本地（）

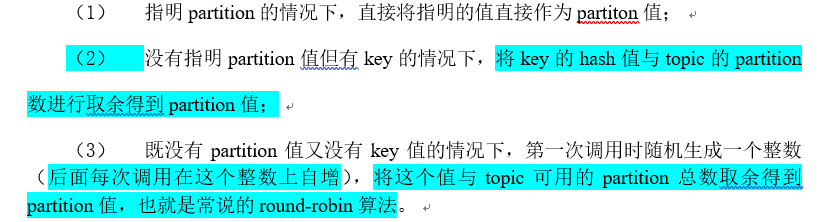
0.9版本之前offset是存放在zk中后来改为存储在本地kafaka的某个主题里原因：消费者本身是和kafka连接在获取数据的同时还要跟zk连接，消费者是以拉去的模式来获取消息的。拉取的速度太快和zk打交道太频繁。（记住消息存放在磁盘，默认保存7天）

1. 消费者组是逻辑上的一个订阅者（把组当成一个大的消费体，这样做是为了提高并发）（效率最好的情况是消费者组里面的消费者个数和topic的的分区个数相等的时候）
2. 一个 topic 可以分为多个 partition（负载均衡），每个 partition 是一个有序的队列
3. 一个 leader 和若干个 follower（leader和follower不在同一个机器上）
4. kafka创建的副本数不能大于机器的台数，这个hdfs是有区别的，hdfs是只最大的副本数即使及其不够也行，在后面扩机器的时候会相应的增加到你指定的副本数
5. 严重注意，kafka向消息队列中写数据的时候不能保证全局有序性，只能保证分区有序性。
6. kafka的文件存取机制



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。

1. kafka分区的原因：方便在集群中扩展，可以提高并发。
2. 分区的原则



1. 数据可靠性保证：ack(3种)+副本同步策略（半数和全部）+ISR（看同步时间）+hw+leo

|  |  |  |
| --- | --- | --- |
| 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 （kafka选中这个）** | 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 |

1. At Least Once + 幂等性 = Exactly Once，Broker 端会对<PID, Partition, SeqNumber>做缓存，当具有相同主键的消息提交时，Broker 只会持久化一条。 但是 PID 重启就会变化，同时不同的 Partition 也具有不同主键，所以幂等性无法保证跨分区跨会话(会话就是连接)的 Exactly Once。
2. Ack决定数据丢不丢的问题（ack有3个策略），ISR是争对副本同步策略，LEO,HW
3. 分区分配策略：range,roundrobin
4. kafka的高速读写数据：分布式分区（读写并行），顺序磁盘，零复制技术
5. 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。
6. Producer事务：解决单次会话里的问题，保证精准一次性写入磁盘。为了实现跨分区跨会话的事务，需要引入一个全局唯一的 Transaction ID（客户端传进去的），并将 Producer 获得的PID 和Transaction ID 绑定。这样当Producer重启后就可以通过正在进行的Transaction ID 获得原来的 PID。
7. offset的维护：什么东西唯一决定一个offset,由组+主题+分区决定（如果组内某个消费者挂了，其他成员能接着消费）。