1. Topic与Partition

Topic 是存储消息的逻辑概念。

Partition

1. 每个topic可以划分多个分区。至少有一个分区。
2. 相同topic下的不同分区中的消息是不同的。

1. 消息的分发策略

消息存储方式：

key->value

key：可选项

根据key按照分区策略将消息放入指定分区中。

自定义分区策略：

实现org.apache.kafka.clients.producer.Partitioner接口

默认分区算法为：hash取模算法

如key为null采用随机分配算法。

metadata.max.age.ms 在指定时间范围内随机选择分区10分钟更新一次。

1. 消息消费原理

多个消费者如何分配分区(partition)。

消费者可以指定消费某一个分区的消息。

同一groupId的消费者会均匀消费多个分区中的消息。如：topic：test中有三个分区为：P0、P1及P2，有三个消费者C1、C2、C3在同一groupId中消费topic=test中消息。那么消费者将均匀消费P0、P1、P2分区中的消息。如：C0消费分区P1;C1消费分区P0;C2消费分区P2。

分几种情况：

Partition数量 = 消费者数量：均匀消费

Partition数量 < 消费者数量：存在消费者不消费消息

Partition数量 > 消费者数量：存在消费者消费多个分区

同一Partition中不存在并发。

增减consumer、partition、broker会导致Rebalance（重新负载）。

Rebalance->分区分配策略：

重新规划Consumer消费Partition。

Range(范围)-默认方式：

同一Topic多个Partition。对Topic的多个Partition按顺序排序，

假设有10分区，3个消费者。如：

分区：0，1，2，3，4，5，6，7，8，9

消费者：C1,C2,C3

分区数量/消费者,除不尽多消费。

C1:0,1,2,3

C2:4,5,6

C3:7,8,9

RoundBobin(轮询)：

将所有的Partition及Consumer的数量都列出来，按hashCode排序，采用轮询方式分配。

ProducerConfig.partition.assign.strategy:参数指定分区分配策略

什么时候触发Rebalance(分区分配策略)？

1. 对于同一Consumer Group中新增消费者。
2. 消费者离开Consumer Group
3. Topic中新增了Partition
4. 消费者主动取消订阅Topic

谁来执行Rebalance以及管理Consumer Group。

Coordinator

当Consumer Group中的第一个Consumer启动时会与broker通信确定Coordinator。

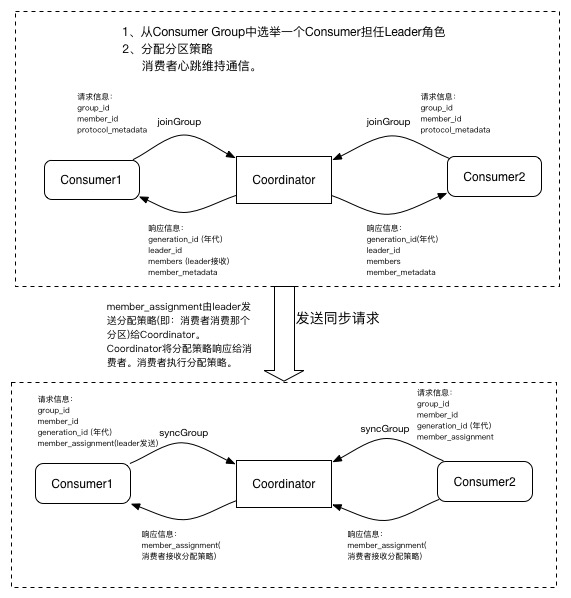
向集群的任意Broker发送GroupCoordinatorRequest请求，服务端返回负载最小的Broker节点

的ID，即：为Coordinator。

JoinGroup：消费者请求Coordinator加入到Consumer Group中。

SyncGroup：

Rebalance机制示意图：



Offset：每个分区中都会有offset，从0开始。消息发送到分区后offset会递增。

Offset存储：\_\_consumer\_offsets对应的Topic 即在zookeeper的 /brokers/toptic

/\_\_consumer\_offsets节点下。

\_\_consumer\_offsets有50个partition。

首先确认指定的Consumer Group存储在哪个分区上。计算方式:

groupId.hashCode % 50 (\_\_consumer\_offsets分区数)

查看\_\_consumer\_offsets Topic的内容：

0.11.0.0之前版本

|  |
| --- |
| sh kafka-simple-consumer-shell.sh --topic \_\_consumer\_offsets –partition 2 –brokers-list 192.168.3.14:9092,192.168.3.36:9092,192.168.3.37:9092 –formatter “kafka.coordinator.group.GroupMetadataManager\$OffsetsMessageFormatter” |

0.11.0.0之后版本

|  |
| --- |
| kafka-console-consumer.sh --topic \_\_consumer\_offsets --partition 4 --bootstrap-server 192.168.3.36:9092 --formatter "kafka.coordinator.group.GroupMetadataManager\$OffsetsMessageFormatter" --consumer.config config/consumer.properties --from-beginning |

Partition的命名方式：Topic名称-递增序号(从0开始)。如：test-0、test-1、test-2。

1. 消息的存储策略

消息保存路径：

topic->partition0

->partition1

->partition2

存放到kafka-logs目录：如test-0、test-1、test-3即：Topic test的分区0、分区1、分区2.

消息写入性能：

按照顺序写入。零拷贝（IO）

消息存储策略:

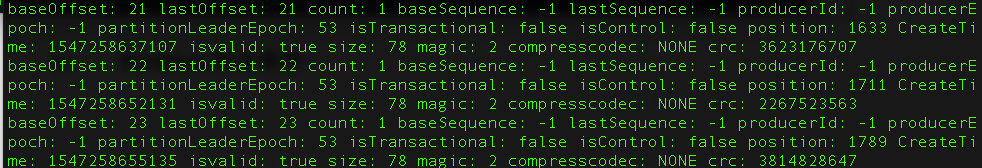
LogSegment 分段保存。文件达到预设值重新生成。

Index->log



查看消息内容：

|  |
| --- |
| sh kafka-run-class.sh kafka.tools.DumpLogSegments --files /tmp/kafka-logs/test-0/00000000000000000000.log –print-data-log |



LogSegment 分段保存。大小设置在：server.properties中的log.segment.bytes 默认为：1GB。

|  |
| --- |
| baseOffset: 35 lastOffset: 35 count: 1 baseSequence: -1 lastSequence: -1 producerId: -1 producerEpoch: -1 partitionLeaderEpoch: 34 isTransactional: false isControl: false position: 2725 CreateTime: 1547258646121 isvalid: true size: 78 magic: 2 compresscodec: NONE crc: 1583354190 |

消息内容：

索引文件与日志文件的对应关系：

日志的清理和压缩策略：

清理策略：

根据时间：消息的保存时间，默认为：7天。配置参数：log.retention.hours=168

根据日志文件大小。配置参数：log.retention.bytes=1073741824

压缩策略：

根据相同的KEY取最新的值。

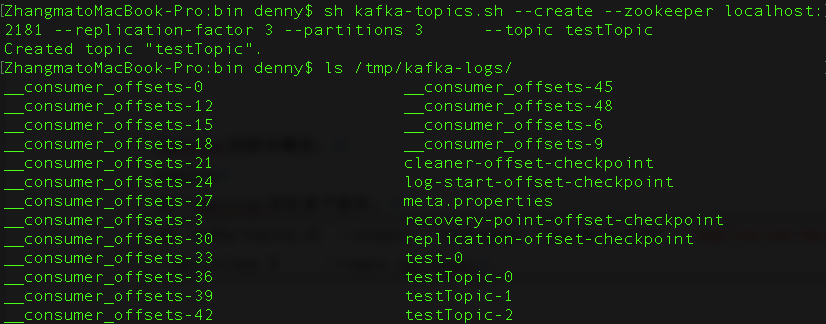
1. Partition副本策略

Partiton的副本概念：

replica

同一Partiton存在多个副本。

|  |
| --- |
| sh kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 3 --topic testTopic |



如果ISR为空怎么办?并且leader副本也挂了。

处理策略：

1. 等待ISR中的任意一个replica活过来，重新选举leader。
2. 选择第一个活过来的replica作为leader。

ISR的设计思想：

同步复制阻塞响应慢，所有副本同步成功后才能响应，如果任一副本宕机就会导致HW不能升级导致无法消费消息。

异步复制存在数据丢失。不管同步是否成功都会反馈。如果同步中follower宕机，消息就会丢失。

ISR只选择副本集群中合适的副本集，这样副本延迟不高，副本宕机就会踢出。后续HW的递增不需要踢出的副本参与确认。

Kafka监控工具：

Kafka monitor、kafka offset monitor、kafka-manager。

分区应该设置多少合适？

根据吞吐量设置。

* 采用操作系统层面的页缓存来存储数据。
* 日志采用顺序写入以及零拷贝的方式提升IO性能。
* Partition的水平分区的概念，能够把一个topic拆分多个分区。
* 发送端和消费端都可以采用并行的方式来消费分区中的消息。