记得录屏

kafka 的版本差异

# 安装kafka

1. tar -zxvf .
2. 进入到config目录下修改server.properties

broker.id

listeners=PLAINTEXT://192.168.11.140:9092

zookeeper.connect

1. 启动

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

sh kafka-server-stop.sh

**zookeeper上注册的节点信息**

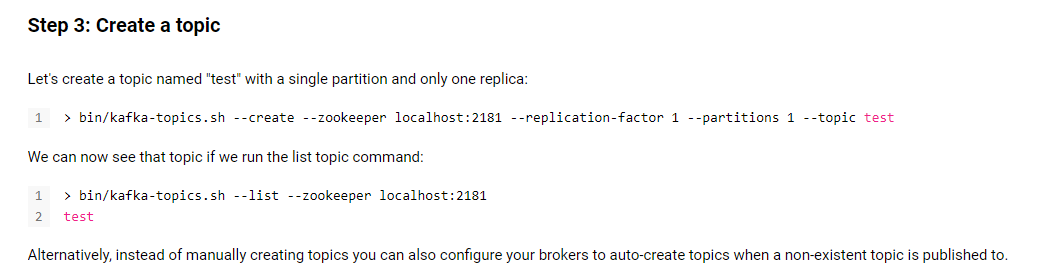
cluster, controller, controller\_epoch, brokers, zookeeper, admin, isr\_change\_notification, consumers, latest\_producer\_id\_block, config

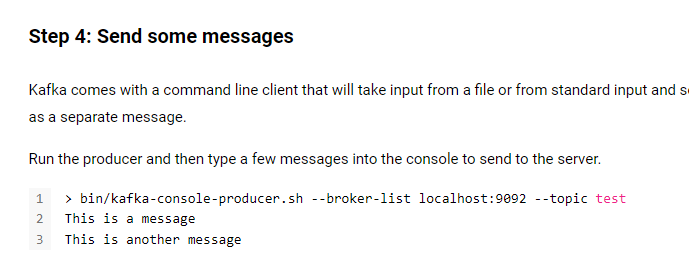
controller – 控制节点

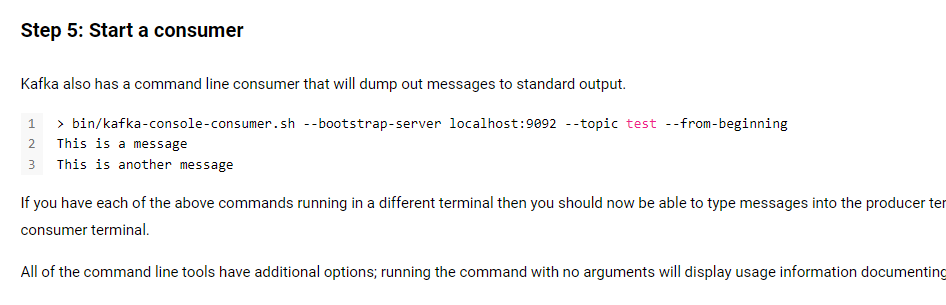
brokers – kafka集群的broker信息 。 topic

consumer ids/owners/offsets

# 基本操作







http://kafka.apache.org/documentation/#quickstart

# kafka的实现细节

## 消息

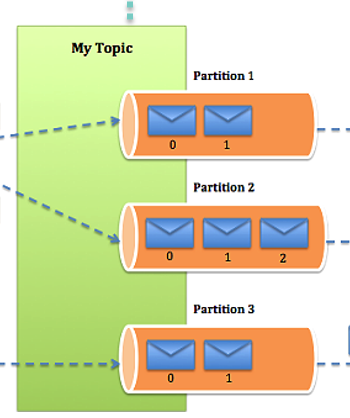
消息是kafka中最基本的数据单元。消息由一串字节构成，其中主要由key和value构成，key和value也都是byte数组。key的主要作用是根据一定的策略，将消息路由到指定的分区中，这样就可以保证包含同一key的消息全部写入到同一个分区中，key可以是null。为了提高网络的存储和利用率，生产者会批量发送消息到kafka，并在发送之前对消息进行压缩

## topic&partition

Topic是用于存储消息的逻辑概念，可以看作一个消息集合。每个topic可以有多个生产者向其推送消息，也可以有任意多个消费者消费其中的消息

每个topic可以划分多个分区（每个Topic至少有一个分区），同一topic下的不同分区包含的消息是不同的。每个消息在被添加到分区时，都会被分配一个offset（称之为偏移量），它是消息在此分区中的唯一编号，kafka通过offset保证消息在分区内的顺序，offset的顺序不跨分区，即kafka只保证在同一个分区内的消息是有序的；

图片包含 文字

已生成高可信度的说明  

Partition是以文件的形式存储在文件系统中，存储在kafka-log目录下，命名规则是：<topic\_name>-<partition\_id>

## kafka的高吞吐量的因素

1. 顺序写的方式存储数据 ；
2. 批量发送；在异步发送模式中。kafka允许进行批量发送，也就是先讲消息缓存到内存中，然后一次请求批量发送出去。这样减少了磁盘频繁io以及网络IO造成的性能瓶颈

batch.size 每批次发送的数据大小

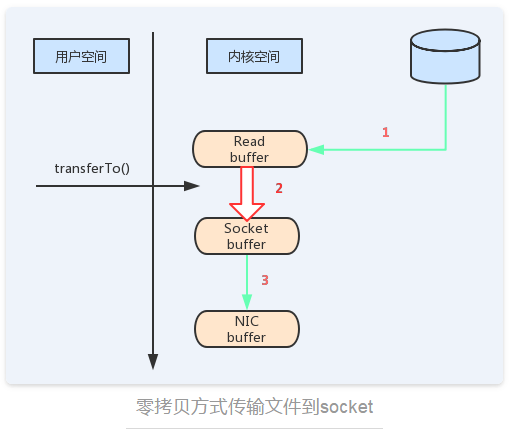
linger.ms 间隔时间

1. 零拷贝

消息从发送到落地保存，broker维护的消息日志本身就是文件目录，每个文件都是二进制保存，生产者和消费者使用相同的格式来处理。在消费者获取消息时，服务器先从硬盘读取数据到内存，然后把内存中的数据原封不懂的通过socket发送给消费者。虽然这个操作描述起来很简单，但实际上经历了很多步骤

|  |  |
| --- | --- |
| ▪ 操作系统将数据从磁盘读入到内核空间的页缓存  ▪ 应用程序将数据从内核空间读入到用户空间缓存中  ▪ 应用程序将数据写回到内核空间到socket缓存中  ▪ 操作系统将数据从socket缓冲区复制到网卡缓冲区，以便将数据经网络发出 | http://rdcqii.hundsun.com/portal/data/upload/201705/f_ea3eaa588e7f176aa71161e0fd08de9e.png |

通过“零拷贝”技术可以去掉这些没必要的数据复制操作，同时也会减少上下文切换次数



## 日志策略

### 日志保留策略

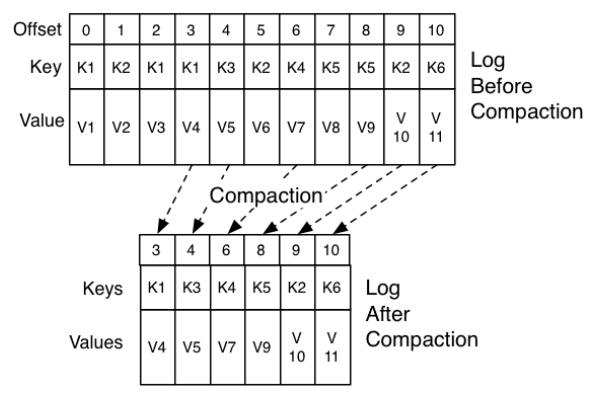
无论消费者是否已经消费了消息，kafka都会一直保存这些消息，但并不会像数据库那样长期保存。为了避免磁盘被占满，kafka会配置响应的保留策略（retention policy），以实现周期性地删除陈旧的消息

kafka有两种“保留策略”：

1. 根据消息保留的时间，当消息在kafka中保存的时间超过了指定时间，就可以被删除；
2. 根据topic存储的数据大小，当topic所占的日志文件大小大于一个阀值，则可以开始删除最旧的消息

### 日志压缩策略

在很多场景中，消息的key与value的值之间的对应关系是不断变化的，就像数据库中的数据会不断被修改一样，消费者只关心key对应的最新的value。我们可以开启日志压缩功能，kafka定期将相同key的消息进行合并，只保留最新的value值



# 消息可靠性机制

## 消息发送可靠性

生产者发送消息到broker，有三种确认方式（request.required.acks）

acks = 0: producer不会等待broker（leader）发送ack 。因为发送消息网络超时或broker crash(1.Partition的Leader还没有commit消息 2.Leader与Follower数据不同步)，既有可能丢失也可能会重发。

acks = 1: 当leader接收到消息之后发送ack，丢会重发，丢的概率很小

acks = -1: 当所有的follower都同步消息成功后发送ack. 丢失消息可能性比较低。

## 消息存储可靠性

每一条消息被发送到broker中，会根据partition规则选择被存储到哪一个partition。如果partition规则设置的合理，所有消息可以均匀分布到不同的partition里，这样就实现了水平扩展。

在创建topic时可以指定这个topic对应的partition的数量。在发送一条消息时，可以指定这条消息的key，producer根据这个key和partition机制来判断这个消息发送到哪个partition。

kafka的高可靠性的保障来自于另一个叫副本（replication）策略，通过设置副本的相关参数，可以使kafka在性能和可靠性之间做不同的切换。

**高可靠性的副本**

**sh kafka-topics.sh --create --zookeeper 192.168.11.140:2181 --replication-factor 2 --partitions 3 --topic sixsix**

**--replication-factor表示的副本数**

## 副本机制

ISR（副本同步队列）

维护的是有资格的follower节点

1. 副本的所有节点都必须要和zookeeper保持连接状态
2. 副本的最后一条消息的offset和leader副本的最后一条消息的offset之间的差值不能超过指定的阀值，这个阀值是可以设置的（replica.lag.max.messages）

## HW&LEO

关于follower副本同步的过程中，还有两个关键的概念，HW(HighWatermark)和LEO(Log End Offset). 这两个参数跟ISR集合紧密关联。HW标记了一个特殊的offset，当消费者处理消息的时候，只能拉去到HW之前的消息，HW之后的消息对消费者来说是不可见的。也就是说，取partition对应ISR中最小的LEO作为HW，consumer最多只能消费到HW所在的位置。每个replica都有HW，leader和follower各自维护更新自己的HW的状态。对于leader新写入的消息，consumer不能立刻消费，leader会等待该消息被所有ISR中的replicas同步更新HW，此时消息才能被consumer消费。这样就保证了如果leader副本损坏，该消息仍然可以从新选举的leader中获取

LEO 是所有副本都会有的一个offset标记，它指向追加到当前副本的最后一个消息的offset。当生产者向leader副本追加消息的时候，leader副本的LEO标记就会递增；当follower副本成功从leader副本拉去消息并更新到本地的时候，follower副本的LEO就会增加