Kafka

王晖

1 Kafka概述

* 1. 定义

Kafka是一个分布式、分区、多副本的基于发布/订阅模式、zookeeper协调的消息队列（Message Queue），主要应用于常见可以用于web/nginx日志、访问日志、消息服务、大数据实时处理等等。

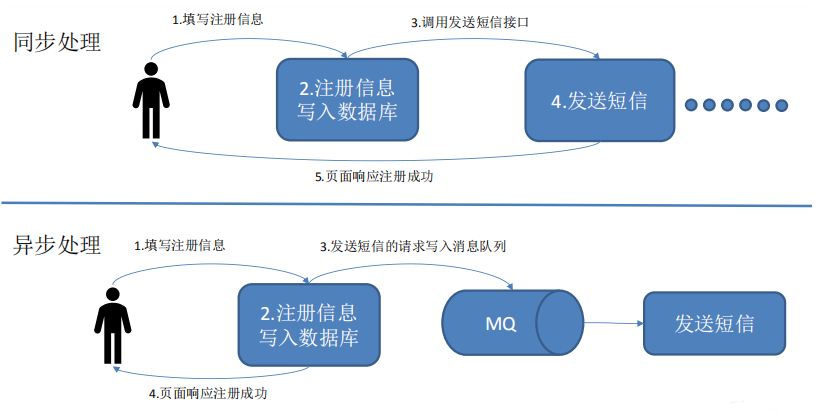
- 可扩展性：kafka集群支持热扩展

- 持久性、可靠性：消息被持久化到磁盘，支持数据备份防止数据丢失

- 容错性：允许集群中节点失败（若副本数量为n,则允许n-1个节点失败）

- 高并发：支持数千个客户端同时读写

* 1. 消息队列
     1. 传统消息队列的应用场景



1. 使用消息队列的好处

1）解耦

消息系统在处理过程中间插入了一个隐含的、基于数据的接口层，两边的处理过程都要实现这一接口。允许你独立的扩展或修改两边的处理过程，只要确保它们遵守同样的接口约束。

2）可恢复性

系统的一部分组件失效时，不会影响到整个系统。消息队列降低了进程间的耦合度，所以即使一个处理消息的进程挂掉，加入队列中的消息仍然可以在系统恢复后被处理。

3）缓冲

有助于控制和优化数据流经过系统的速度，解决生产消息和消费消息的处理速度不一致的情况。

1. 峰值处理能力（削峰）& 灵活

在访问量剧增的情况下，应用仍然需要继续发挥作用，但是这样的突发流量并不常见。如果为以能处理这类峰值访问为标准来投入资源随时待命无疑是巨大的浪费。使用消息队列能够使关键组件顶住突发的访问压力，而不会因为突发的超负荷的请求而完全崩溃。分布式架构也可根据峰值需求灵活增减服务器数量

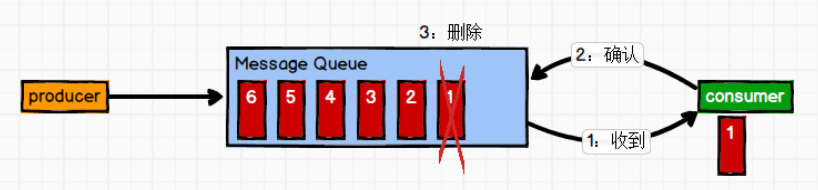
5）异步通信

很多时候，用户不想也不需要立即处理消息。消息队列提供了异步处理机制，允许用户把一个消息放入队列，但并不立即处理它。先向队列中放入消息，然后在需要的时候再去处理它们。

* + 1. 消息队列的两种模式

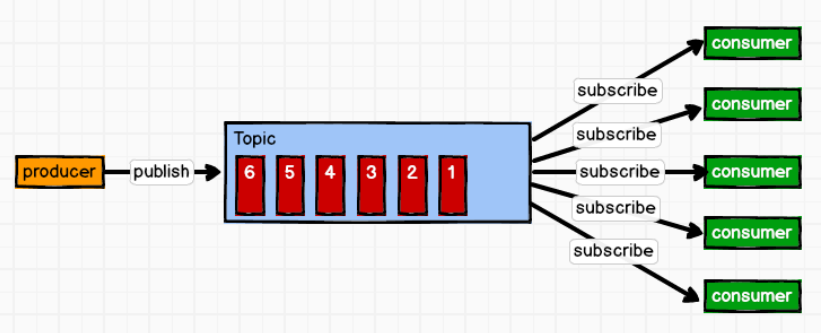
1. 点对点模式（一对一，消费者主动拉取数据，消息收到后消息清除）

消息生产者生产消息发送到Queue中，然后消息消费者从 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。

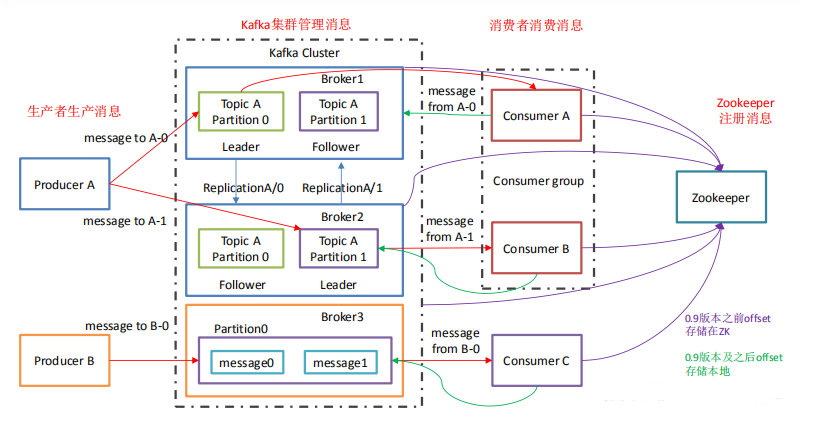


1. 发布/订阅模式（一对多，消费者消费数据之后不会清除消息）

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。



* 1. 基础架构



**1）Producer ：**消息生产者，就是向kafka broker发消息的客户端；

**2）Consumer ：**消息消费者，向kafka broker取消息的客户端；

**3）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组内消费者消费；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**。

**4）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。

**5）Topic ：主题**可以理解为一个队列，**生产者和消费者面向的都是一个topic**；

**6）Partition：分区** 为了实现扩展性，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。注意：一分区不能同时被同一CG里的多个C消费。

**7）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**。

**8）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。

**9）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。

1. Kafka安装部署
   1. 安装部署
      1. 集群规划

示例：

|  |  |  |  |
| --- | --- | --- | --- |
| 主机名 | IP | 系统 | 安装内容 |
| ELK1 | 10.145.210.177 | Centos7 | Elasticsearch/zookeeper/kafka/  logstash/kibana |
| ELK2 | 10.145.210.178 | Centos7 | Elasticsearch/zookeeper/kafka |
| ELK3 | 10.145.210.179 | Centos7 | Elasticsearch/zookeeper/kafka |
| logcenter | 10.145.210.174 | Centos7 | Filebeat |

* + 1. zookeeper安装

1）Kafka压缩包中集成了zookeeper。

#tar -xzvf /home/paasadmin/kafka\_2.12-2.2.0.tgz -C /usr/local

2）修改zookeeper配置

#cd /usr/local/kafka\_2.12-2.2.0/config

#vim zookeeper.properties

dataDir=/opt/data/zookeeper/data

dataLogDir=/opt/data/zookeeper/logs

clientPort=2166 #客户端连接的接口

tickTime=2000 #CS通信心跳时间（毫秒）

initLimit=20 #FL初始连接时能容忍的最多心跳数

syncLimit=10

#集群信息的配置

server.13 = 10.145.210.177:2666:3666

server.14 = 10.145.210.178:2666:3666

server.15 = 10.145.210.179:2666:3666

1. 配置相应文件和zookeeper-id

##创建配置文件中的zookeeper数据和日志文件夹

#mkdir -p /opt/data/zookeeper/{data,logs}

##在三台zookeeper主机上配置myid，对应配置文件

10.145.210.177#echo 13 > /opt/data/zookeeper/data/myid

10.145.210.178#echo 14 > /opt/data/zookeeper/data/myid

10.145.210.179#echo 15 > /opt/data/zookeeper/data/myid

* + 1. kafka安装

1）修改kafka配置文件

#mv server.properties server.properties.bak && vim server.properties

**#broker的全局唯一编号，不能重复**

**broker.id** = **1**

**#用来监听链接的地址和端口，producer或consumer将在此端口建立连接**

**listeners=PLAINTEXT://**10.145.210.177**:6230**

#kafka运行日志存放的路径

**log.dirs =** /opt/data/kafka/logs

#连接zk

**zookeeper.connect** = 10.145.210.177**:2166,**10.145.210.178**:2166,**10.145.210.179**:2166**

num.network.threads = 18 #处理网络请求的线程数量，也就是接收消息的线程数。

num.io.threads = 30 #用来处理磁盘IO的线程数量

#发送、接受、请求套接字的缓冲区大小

socket.send.buffer.bytes = 102400

socket.receive.buffer.bytes = 102400

socket.request.max.bytes = 104857600

num.partitions = 30　 #topic在当前broker上的分片个数

num.recovery.threads.per.data.dir = 1 #用来设置恢复和清理data数据的线程数

#kafka集群中 topic的偏移量和节点的副本数

offsets.topic.replication.factor = 1

transaction.state.log.replication.factor = 1

transaction.state.log.min.isr = 1

default.replication.factor = 1

#kafka日志管理

log.retention.hours = 72

log.segment.bytes = 1073741824

log.retention.check.interval.ms = 300000

log.flush.interval.messages = 20000

log.flush.interval.ms = 1000

zookeeper.connection.timeout.ms = 6000

auto.create.topics.enable = true

#可以从非ISR集合中选举follower副本称为新的leader，介绍ISR后详解

unclean.leader.election.enable = true

1. 配置相应文件

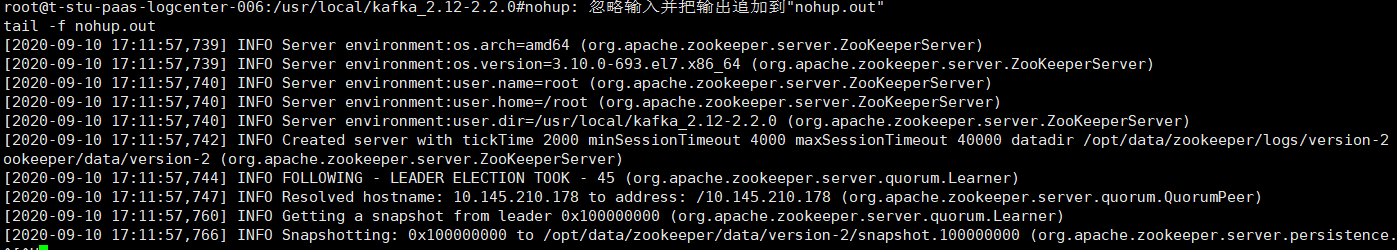
#mkdir -p /opt/data/kafka/logs

* 1. 命令行实操

1. 集群启动

#cd /usr/local/kafka\_2.12-2.2.0

#nohup bin/kafka-server-start.sh config/server.properties &



1. 查看当前服务器中的所有 topic

#cd /usr/local/kafka\_2.12-2.2.0

#bin/kafka-topics.sh --zookeeper localhost:2166 --**list**

1. 创建topic

#/usr/local/kafka\_2.12-2.2.0/bin/kafka-topics.sh --create --zookeeper localhost:2166 --replication-factor 3 --partitions 3 --topic testtopic

选项说明：

--topic 定义topic名

--replication-factor定义副本数

--partitions 定义分区数

1. 查看topic

##查询10.145.210.177上的topic

#/usr/local/kafka\_2.12-2.2.0/bin/kafka-topics.sh --zookeeper 10.145.210.177:2166 --list

##查询10.145.210.178上的topic

#/usr/local/kafka\_2.12-2.2.0/bin/kafka-topics.sh --zookeeper 10.145.210.178:2166 --list

##查询10.145.210.179上的topic

#/usr/local/kafka\_2.12-2.2.0/bin/kafka-topics.sh --zookeeper 10.145.210.179:2166 --list



1. 删除topic

##需要在server.properties 中设置 delete.topic.enable=true 否则只是将标记删除。

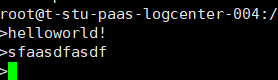
#/usr/local/kafka\_2.12-2.2.0/bin/kafka-topics.sh --zookeeper 10.145.210.177:2166 --delete --topic testtopic

1. 模拟生产和消费收发消息

##从10.145.210.177发送消息

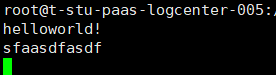
#/usr/local/kafka\_2.12-2.2.0/bin/kafka-console-producer.sh --broker-list 10.145.210.177:6230 --topic testtopic

>hello world



##从10.145.210.178上接收消息

#/usr/local/kafka\_2.12-2.2.0/bin/kafka-console-consumer.sh --bootstrap-server 10.145.210.178:6230 --topic testtopic --from-beginning



--from-beginning：会把主题中以往所有的数据都读取出来

1. 查看某个topic的详情

#/usr/local/kafka\_2.12-2.2.0/bin/kafka-topics.sh --zookeeper 10.145.210.177:2166 --describe --topic testtopic

1. 修改分区数

#/usr/local/kafka\_2.12-2.2.0/bin/kafka-topics.sh --zookeeper 10.145.210.177:2166 --alter --topic testtopic partitions 6

* 1. Kafka-manager

2.3.1 安装部署

#unzip kafka-manager-2.0.0.2.zip

#cd /usr/local/kafka-manager-2.0.0.2 && vim conf/application.conf

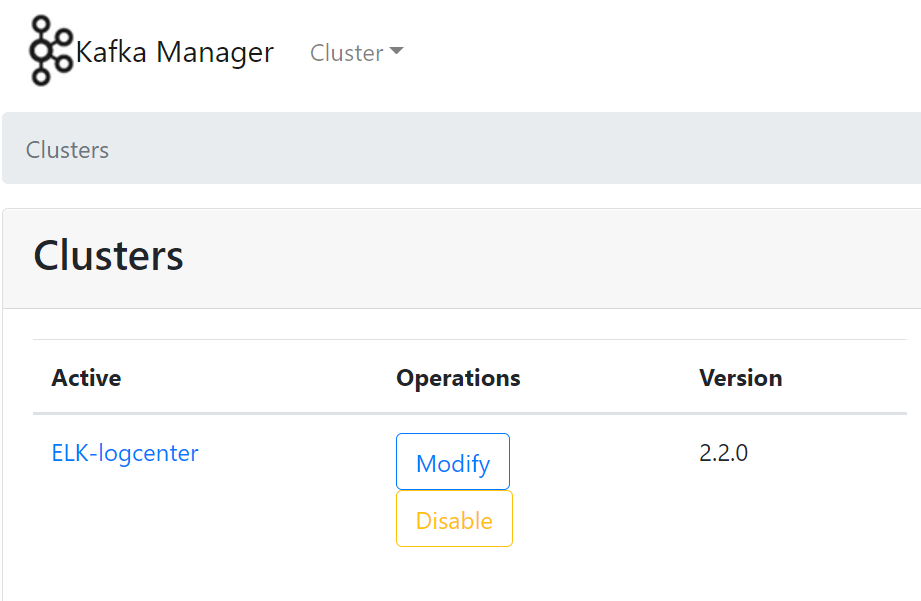
#修改kafka-manager.zkhosts为

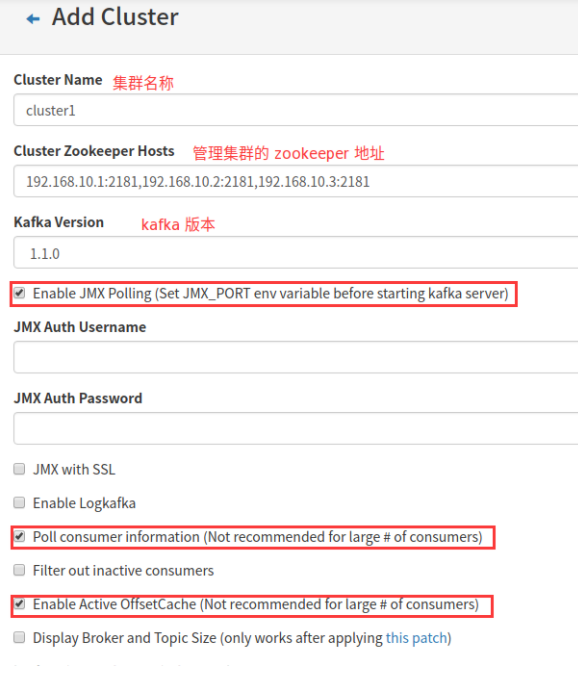
kafka-manager.zkhosts="10.145.210.177:2166,10.145.210.178:2166,10.145.210.179:2166"

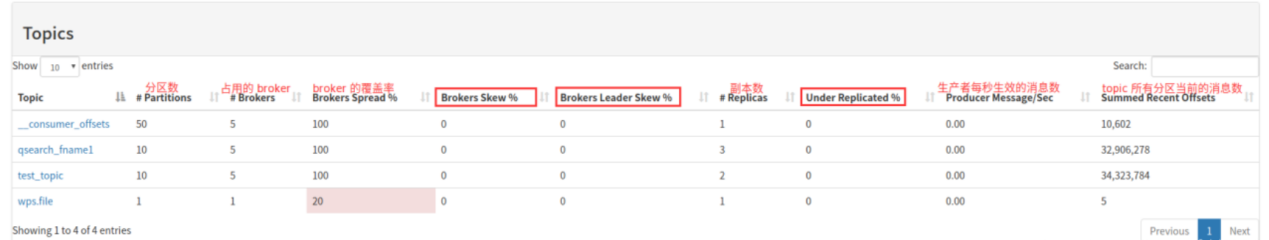
2.3.2 启动测试

#cd /usr/local/kafka-manager-2.0.0.2 && nohup ./bin/kafka-manager &

浏览器进入http://10.145.210.177:9000查看



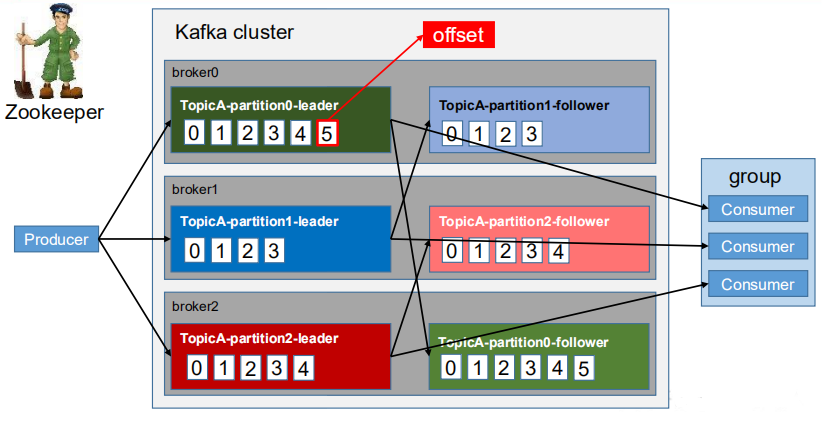




3 Kafka架构深入

3.1 Kafka工作流程及文件存储机制

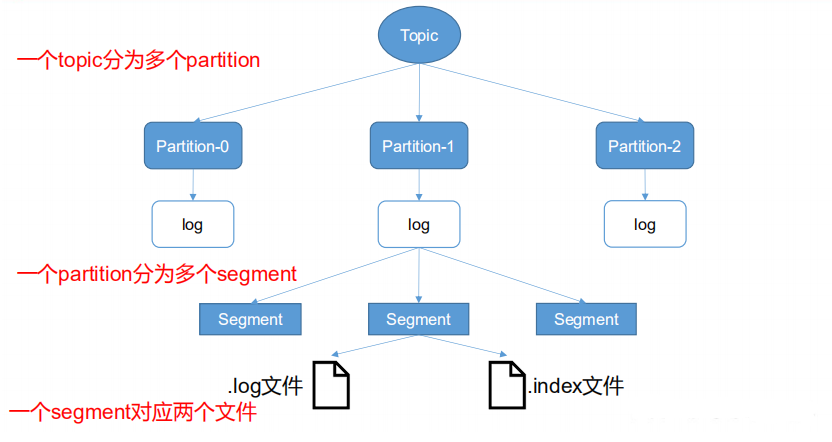
3.1.1 工作流程



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。

3.1.2 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，first 这个 topic 有三个分区，则其对应的文件夹为 first-0,first-1,first-2。

00000000000000000000.index

00000000000000000000.log

00000000000000170410.index

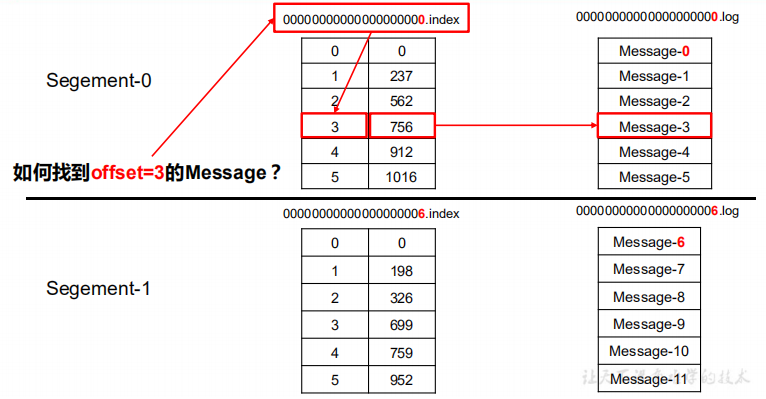
00000000000000170410.log

00000000000000239430.index

00000000000000239430.log

“.index”文件存储索引信息，“.log”文件存储数据，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。

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。



3.2 Kafka生产者

3.2.1 分区策略

1）**分区的目的**

（1）实现了负载均衡和水平扩展，从数据组织形式来说，kafka有三层形式，kafka有多个主题，每个主题有多个分区，每个分区又有多条消息，而每个分区可以分布到不同的机器上，这样整个集群就可以适应任意大小的数据了；

（2）可以提高并发，因为可以以Partition为单位读写了。

2）**分区的设定**

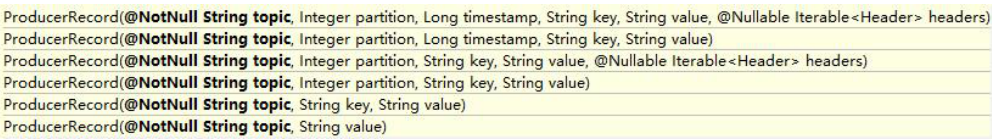
我们可以通过partitions指定创建topic时候所创建的分区数

#bin/kafka-topics.sh --create --bootstrap-server localhost:2166 --replication-factor 3 --partitions 3 --topic testtopic

**分区个数选择：**分区越多，所需要消耗的资源就越多，需要结合具体业务确定分区个数，假如每秒钟需要从主题写入和读取1GB数据，而消费者1秒钟最多处理50MB的数据，那么这个时候就可以设置20-25个分区，当然还要结合具体的物理资源情况。如何无法估算出大概的处理速度和时间，可设为broker机器数量的2~3倍。

3）**分区的原则**

我们需要将 producer 发送的数据封装成一个 ProducerRecord对象。



topic字段用于指定该消息记录的一级分属；partition字段和键key字段用于指定该消息记录的二级分属；value字段用于存放该消息记录的有效载荷；timestamp字段用于指定时间戳；

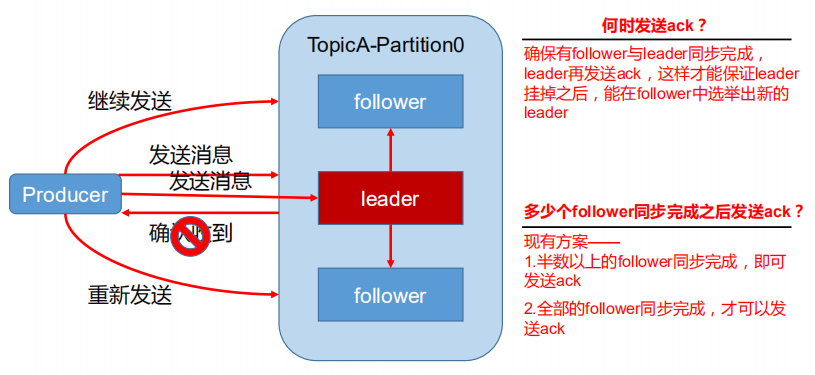
（1）指明partition的情况下，直接将指明的值直接作为partiton值；

（2）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；

（3）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。

3.2.2 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。



1. **kafka副本机制**

在kafka中，每个主题可以有多个分区，每个分区又可以有多个副本。这多个副本中，只有一个是leader，而其他的都是follower副本。通过这样的机制实现了高可用，当某台机器挂掉后，其他follower副本也能迅速“转正”，开始对外提供服务。

在kafka中，实现副本的目的就是冗余备份，且仅仅是冗余备份，所有的读写请求都是由leader副本进行处理的。follower副本仅有一个功能，那就是从leader副本拉取消息，尽量让自己跟leader副本的内容一致。本质上是对性能和数据一致性的取舍。

kafka分区副本的选举与zookeeper是类似的，都是选择最新的那个follower副本，但它是通过一个In-sync（ISR）副本集合实现。

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 |

Kafka 倾向于第二种方案，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。

2.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。

1. **ISR （**in-sync replica set**）**

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。

所以kafka的完全同步指的是Kafka只保证对ISR集合中的所有副本保证完全同步

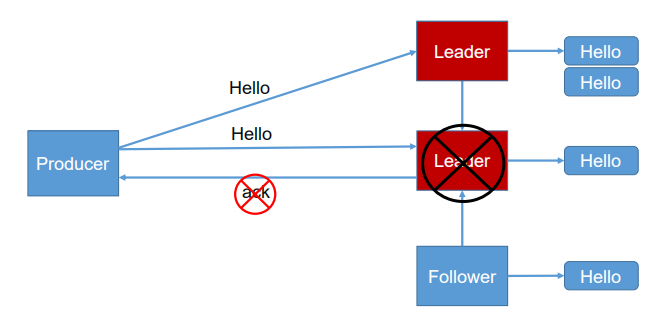
1. ack应答机制

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。ack:

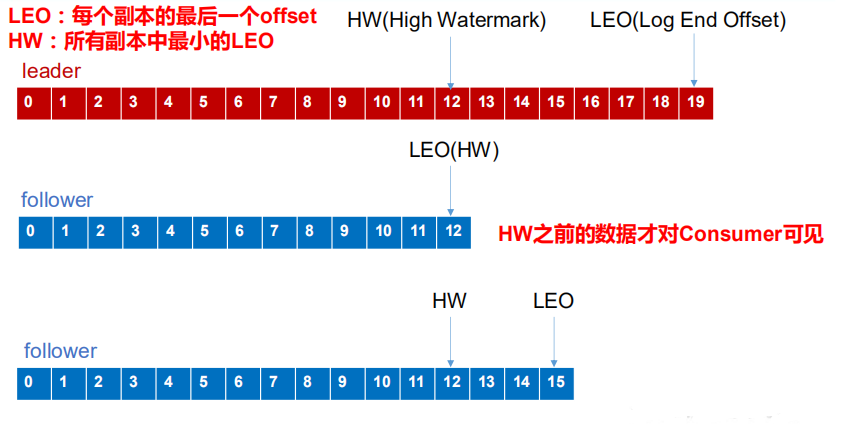
①acks=0，生产者在成功写入消息之前不会等待服务器的响应。埋头写，速度最快，**消息丢失**的话生产者无从得知。

②acks=1，只要集群的首领节点收到消息，生产者就会收到一个来自服务器的成功响应。如果消息无法到达首领节点（比如首领节点崩溃，新的首领还没有被选举出来），生产者会收到一个错误响应，为了避免数据丢失，生产者会重发消息。不过，如果一个没有收到消息的节点被选举为为新首领，消息还是会**丢失**。

③acks=all，所有参与复制的节点全部收到消息时，生产者才会收到一个来自服务器的成功响应，这种模式是最安全的。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**。



1. 故障处理细节



LEO：指的是每个副本最大的 offset；

HW：指的是消费者能见到的最大的offset，ISR队列中最小的LEO。

（1）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。

（2）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：这只能保证副本间的数据一致性，不能保证数据不丢失或者不重复。

3.2.3 Exactly Once语义

ACK = -1(all)时，可以保证Producer到Server之间不会丢失数据，但是不能保证数据不重复，即At Least Once语义。

相对的，ACK= 0时，可以保证生产者每条消息只会被发送一次，但是不能保证数据不丢失，即At Most Once语义。

但是对于一些非常重要的信息，比如说交易数据，下游数据消费者要求数据既不重复也不丢失，即Exactly Once语义。0.11 版本的 Kafka，引入了一项重大特性：幂等性。所谓的幂等性就是指Producer不论向Server发送多少次重复数据，Server端都只会持久化一条。幂等性结合At Least Once语义，就构成了Kafka 的Exactly Once语义。即：

At Least Once + 幂等性 = Exactly Once

要启用幂等性，只需要将Producer的参数中enable.idompotence设置为true 即可。开启幂等性的Producer在初始化的时候会被分配一个PID，发往同一 Partition的消息会附带Sequence Number。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。

3.3 Kafka消费者

1）消费者对象——要从一个Kafka主题读取消息井验证这些消息，然后再把它们保存起来。生产者需要用序列化器把对象转换成字节数组再发送给Kafka，类似地，消费者需要用反序列化器把从Kafka收到的字节转换成Java对象

2）消费者群组—— 一个群组里的消费者订阅的是同一个主题，每个消费者接收主题一部分分区的消息。如果消费者数量>分区数量，一部分消费者会被闲置。 群组之间接受数据互不影响。

3.3.1 消费方式

Kafka最初考虑的问题是，customer应该从brokes拉取消息还是brokers将消息推送到consumer，也就是pull还push。在这方面，Kafka遵循了一种大部分消息系统共同的传统的设计：producer将消息推送到broker，consumer从broker拉取消息。

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。

Pull（拉）模式则可以根据consumer的消费能力以适当的速率消费消息。不过pull有个缺点是，如果broker没有可供消费的消息，将导致consumer不断在循环中轮询并返回空数据，直到新消息到达。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。

3.3.2 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。

1. RoundRobin（轮询）

按消费者组划分，将来自不同主题的消息统一考虑



1. Range （默认）

按主题划分开来单独考虑，随着Topic的增多容易造成负载不均衡



3.3.3 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。

3.4 Kafka高效读写数据

1）**顺序写磁盘**

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 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。

2）**零复制技术**

kafka中的消费者在读取服务端的数据时，需要将服务端的磁盘文件通过网络发送到消费者进程，网络发送需要经过几种网络节点。

传统的读取文件数据并发送到网络的步骤如下：

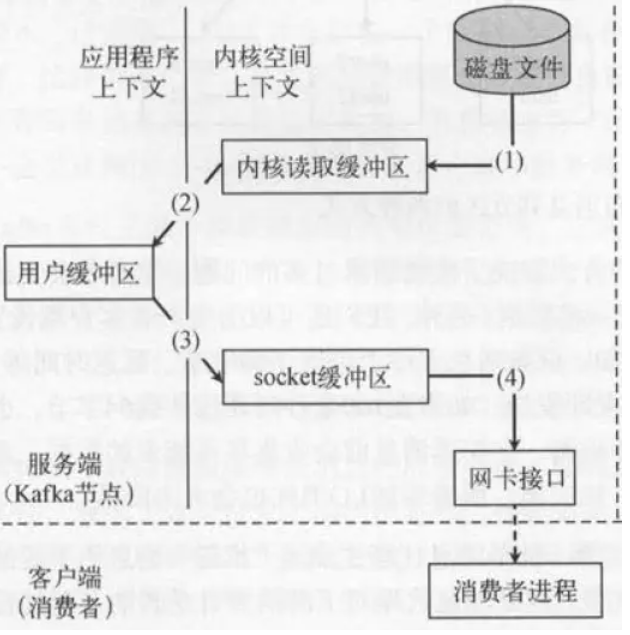
（1）操作系统将数据从磁盘文件中读取到内核空间的页面缓存；

（2）应用程序将数据从内核空间读入用户空间缓冲区；

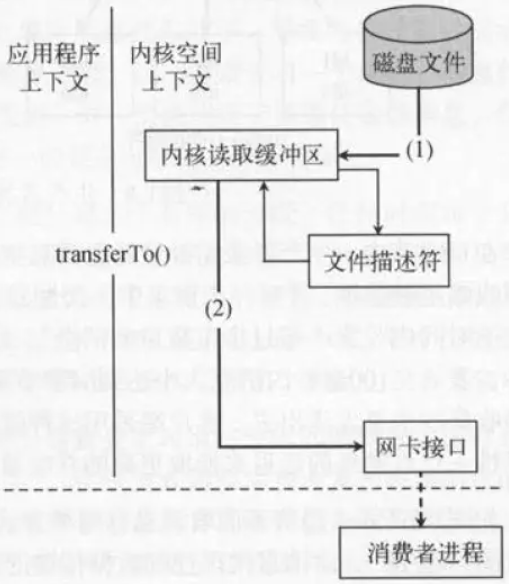
（3）应用程序将读到数据写回内核空间并放入socket缓冲区；

（4）操作系统将数据从socket缓冲区复制到网卡接口，此时数据才能通过网络发送。

这个过程涉及到4次上下文切换以及4次数据的复制，并且有两次复制操作是由CPU完成。但是这个过程中，数据完全没有进行变化，仅仅是从磁盘复制到网卡缓冲区。 如下图所示



通常情况下，Kafka的消息会有多个订阅者，生产者发布的消息会被不同的消费者多次消费，为了优化这个流程，Kafka使用了“零拷贝技术”，如下图所示：

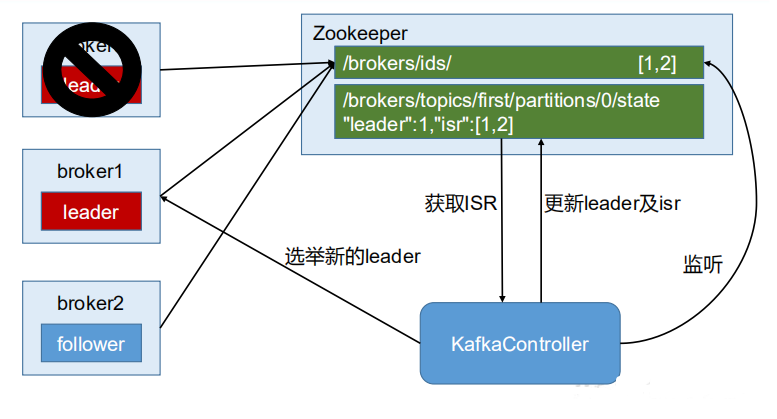


“零拷贝技术”只用将磁盘文件的数据复制到页面缓存中一次，然后将数据从页面缓存直接发送到网络中（发送给不同的订阅者时，都可以使用同一个页面缓存），避免了重复复制操作。

如果有10个消费者，传统方式下，数据复制次数为4\*10=40次，而使用“零拷贝技术”只需要1+10=11次，一次为从磁盘复制到页面缓存，10次表示10个消费者各自读取一次页面缓存。

3.5 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：



3.6 Kafka事务

Kafka从0.11版本开始引入了事务支持。事务可以保证Kafka在Exactly Once 语义的基础上，生产和消费可以跨分区和会话，要么全部成功，要么全部失败。

3.6.1 Producer事务

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，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 TransactionID获得原来的PID。

为了管理 Transaction，Kafka 引入了一个新的组件 Transaction Coordinator。Producer 就是通过和 Transaction Coordinator 交互获得 Transaction ID 对应的任务状态。Transaction Coordinator 还负责将事务所有写入 Kafka 的一个内部 Topic，这样即使整个服务重启，由于事务状态得到保存，进行中的事务状态可以得到恢复，从而继续进行。

3.6.2 Consumer事务

上述事务机制主要是从 Producer 方面考虑，对于 Consumer 而言，事务的保证就会相对较弱，尤其时无法保证 Commit 的信息被精确消费。这是由于 Consumer 可以通过 offset 访问任意信息，而且不同的 Segment File 生命周期不同，同一事务的消息可能会出现重启后被删除的情况。