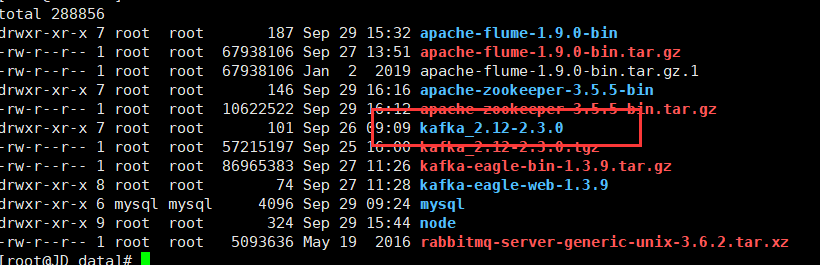
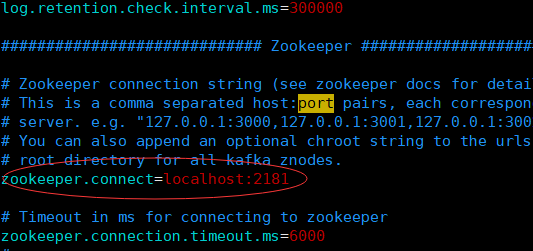
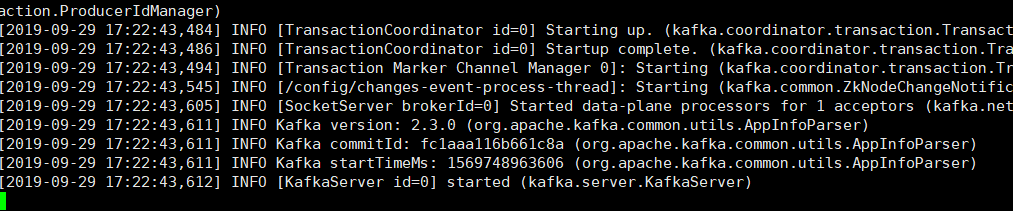
# Kafka安装配置文档

## Kafka简介

#### Kafka是一个分布式消息队列。★Kafka对消息保存时根据Topic进行归类，发送消息者称为Producer，消息接受者称为Consumer，此外kafka集群有多个kafka实例组成，每个实例(server)称为broker。无论是kafka集群，还是consumer都依赖于zookeeper集群保存一些meta信息，来保证系统可用性。

## Centos7安装

1. 首先要有zookeeper环境
2. 去官网下载解压包[**http://mirrors.tuna.tsinghua.edu.cn/apache/kafka/2.3.0/kafka-2.3.0-src.tgz**](http://mirrors.tuna.tsinghua.edu.cn/apache/kafka/2.3.0/kafka-2.3.0-src.tgz)
3. 然后上传到服务器解压
4. 修改配置文件config/server. Properties中链接zookeeper地址
5. 然后启动bin/kafka-server-start.sh config/server.properties

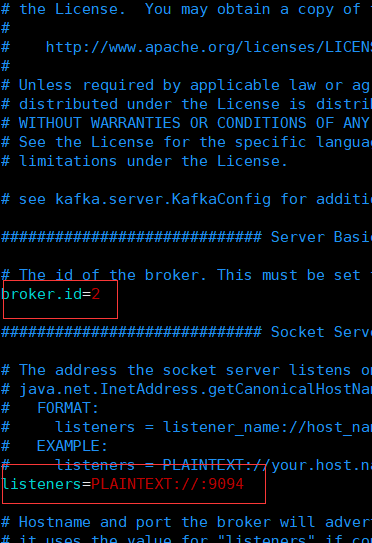
## 伪集群配置

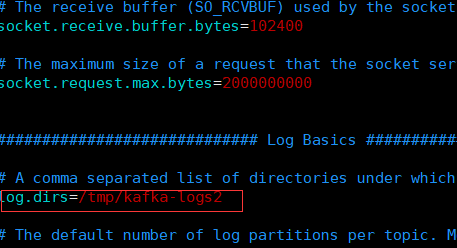
复制一份server.properties出来



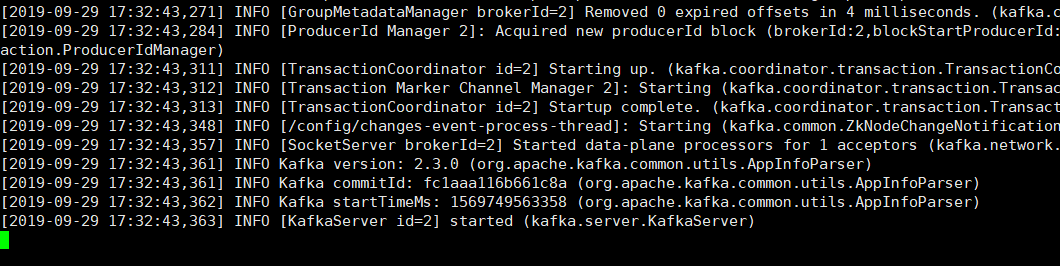
然后修改server2.properties

需要修改里面的三个值





然后启动bin/kafka-server-start.sh config/server3.properties



## 配置

参数

说明(解释)

broker.id =0

每一个broker在集群中的唯一表示，要求是正数。当该服务器的IP地址发生改变时，broker.id没有变化，则不会影响consumers的消息情况

log.dirs=/data/kafka-logs

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

port =9092

broker server服务端口

message.max.bytes =6525000

表示消息体的最大大小，单位是字节

num.network.threads =4

broker处理消息的最大线程数，一般情况下不需要去修改

num.io.threads =8

broker处理磁盘IO的线程数，数值应该大于你的硬盘数

background.threads =4

一些后台任务处理的线程数，例如过期消息文件的删除等，一般情况下不需要去做修改

queued.max.requests =500

等待IO线程处理的请求队列最大数，若是等待IO的请求超过这个数值，那么会停止接受外部消息，应该是一种自我保护机制。

host.name

broker的主机地址，若是设置了，那么会绑定到这个地址上，若是没有，会绑定到所有的接口上，并将其中之一发送到ZK，一般不设置

socket.send.buffer.bytes=100\*1024

socket的发送缓冲区，socket的调优参数SO\_SNDBUFF

socket.receive.buffer.bytes =100\*1024

socket的接受缓冲区，socket的调优参数SO\_RCVBUFF

socket.request.max.bytes =100\*1024\*1024

socket请求的最大数值，防止serverOOM，message.max.bytes必然要小于socket.request.max.bytes，会被topic创建时的指定参数覆盖

log.segment.bytes =1024\*1024\*1024

topic的分区是以一堆segment文件存储的，这个控制每个segment的大小，会被topic创建时的指定参数覆盖

log.roll.hours =24\*7

这个参数会在日志segment没有达到log.segment.bytes设置的大小，也会强制新建一个segment会被 topic创建时的指定参数覆盖

log.cleanup.policy = delete

日志清理策略选择有：delete和compact主要针对过期数据的处理，或是日志文件达到限制的额度，会被 topic创建时的指定参数覆盖

log.retention.minutes=3days

数据存储的最大时间超过这个时间会根据log.cleanup.policy设置的策略处理数据，也就是消费端能够多久去消费数据

log.retention.bytes和log.retention.minutes任意一个达到要求，都会执行删除，会被topic创建时的指定参数覆盖

log.retention.bytes=-1

topic每个分区的最大文件大小，一个topic的大小限制 =分区数\*log.retention.bytes。-1没有大小限log.retention.bytes和log.retention.minutes任意一个达到要求，都会执行删除，会被topic创建时的指定参数覆盖

log.retention.check.interval.ms=5minutes

文件大小检查的周期时间，是否处罚 log.cleanup.policy中设置的策略

log.cleaner.enable=false

是否开启日志压缩

log.cleaner.threads = 2

日志压缩运行的线程数

log.cleaner.io.max.bytes.per.second=None

日志压缩时候处理的最大大小

log.cleaner.dedupe.buffer.size=500\*1024\*1024

日志压缩去重时候的缓存空间，在空间允许的情况下，越大越好

log.cleaner.io.buffer.size=512\*1024

日志清理时候用到的IO块大小一般不需要修改

log.cleaner.io.buffer.load.factor =0.9

日志清理中hash表的扩大因子一般不需要修改

log.cleaner.backoff.ms =15000

检查是否处罚日志清理的间隔

log.cleaner.min.cleanable.ratio=0.5

日志清理的频率控制，越大意味着更高效的清理，同时会存在一些空间上的浪费，会被topic创建时的指定参数覆盖

log.cleaner.delete.retention.ms =1day

对于压缩的日志保留的最长时间，也是客户端消费消息的最长时间，同log.retention.minutes的区别在于一个控制未压缩数据，一个控制压缩后的数据。会被topic创建时的指定参数覆盖

log.index.size.max.bytes =10\*1024\*1024

对于segment日志的索引文件大小限制，会被topic创建时的指定参数覆盖

log.index.interval.bytes =4096

当执行一个fetch操作后，需要一定的空间来扫描最近的offset大小，设置越大，代表扫描速度越快，但是也更好内存，一般情况下不需要搭理这个参数

log.flush.interval.messages=None

log文件”sync”到磁盘之前累积的消息条数,因为磁盘IO操作是一个慢操作,但又是一个”数据可靠性"的必要手段,所以此参数的设置,需要在"数据可靠性"与"性能"之间做必要的权衡.如果此值过大,将会导致每次"fsync"的时间较长(IO阻塞),如果此值过小,将会导致"fsync"的次数较多,这也意味着整体的client请求有一定的延迟.物理server故障,将会导致没有fsync的消息丢失.

log.flush.scheduler.interval.ms =3000

检查是否需要固化到硬盘的时间间隔

log.flush.interval.ms = None

仅仅通过interval来控制消息的磁盘写入时机,是不足的.此参数用于控制"fsync"的时间间隔,如果消息量始终没有达到阀值,但是离上一次磁盘同步的时间间隔达到阀值,也将触发.

log.delete.delay.ms =60000

文件在索引中清除后保留的时间一般不需要去修改

log.flush.offset.checkpoint.interval.ms =60000

控制上次固化硬盘的时间点，以便于数据恢复一般不需要去修改

auto.create.topics.enable =true

是否允许自动创建topic，若是false，就需要通过命令创建topic

default.replication.factor =1

是否允许自动创建topic，若是false，就需要通过命令创建topic

num.partitions =1

每个topic的分区个数，若是在topic创建时候没有指定的话会被topic创建时的指定参数覆盖

以下是kafka中Leader,replicas配置参数

controller.socket.timeout.ms =30000

partition leader与replicas之间通讯时,socket的超时时间

controller.message.queue.size=10

partition leader与replicas数据同步时,消息的队列尺寸

replica.lag.time.max.ms =10000

replicas响应partition leader的最长等待时间，若是超过这个时间，就将replicas列入ISR(in-sync replicas)，并认为它是死的，不会再加入管理中

replica.lag.max.messages =4000

如果follower落后与leader太多,将会认为此follower[或者说partition relicas]已经失效

##通常,在follower与leader通讯时,因为网络延迟或者链接断开,总会导致replicas中消息同步滞后

##如果消息之后太多,leader将认为此follower网络延迟较大或者消息吞吐能力有限,将会把此replicas迁移

##到其他follower中.

##在broker数量较少,或者网络不足的环境中,建议提高此值.

replica.socket.timeout.ms=30\*1000

follower与leader之间的socket超时时间

replica.socket.receive.buffer.bytes=64\*1024

leader复制时候的socket缓存大小

replica.fetch.max.bytes =1024\*1024

replicas每次获取数据的最大大小

replica.fetch.wait.max.ms =500

replicas同leader之间通信的最大等待时间，失败了会重试

replica.fetch.min.bytes =1

fetch的最小数据尺寸,如果leader中尚未同步的数据不足此值,将会阻塞,直到满足条件

num.replica.fetchers=1

leader进行复制的线程数，增大这个数值会增加follower的IO

replica.high.watermark.checkpoint.interval.ms =5000

每个replica检查是否将最高水位进行固化的频率

controlled.shutdown.enable =false

是否允许控制器关闭broker ,若是设置为true,会关闭所有在这个broker上的leader，并转移到其他broker

controlled.shutdown.max.retries =3

控制器关闭的尝试次数

controlled.shutdown.retry.backoff.ms =5000

每次关闭尝试的时间间隔

leader.imbalance.per.broker.percentage =10

leader的不平衡比例，若是超过这个数值，会对分区进行重新的平衡

leader.imbalance.check.interval.seconds =300

检查leader是否不平衡的时间间隔

offset.metadata.max.bytes

客户端保留offset信息的最大空间大小

kafka中zookeeper参数配置

zookeeper.connect = localhost:2181

zookeeper集群的地址，可以是多个，多个之间用逗号分割hostname1:port1,hostname2:port2,hostname3:port3

zookeeper.session.timeout.ms=6000

ZooKeeper的最大超时时间，就是心跳的间隔，若是没有反映，那么认为已经死了，不易过大

zookeeper.connection.timeout.ms =6000

ZooKeeper的连接超时时间

zookeeper.sync.time.ms =2000

ZooKeeper集群中leader和follower之间的同步实际那

————————————————

版权声明：本文为CSDN博主「星辰大海的风景」的原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/zhongwumao/article/details/81171143