kafka 配置文件参数详解

kafka的配置分为 broker、producter、consumer三个不同的配置

一 BROKER 的全局配置

最为核心的三个配置 broker.id、log.dir、zookeeper.connect 。

```
----------------------------------------- 系统 相关 -----------------------------------------
##每一个broker在集群中的唯一标示，要求是正数。在改变IP地址，不改变broker.id的话不会影响consumers
broker.id =1

##kafka数据的存放地址，多个地址的话用逗号分割 /tmp/kafka-logs-1，/tmp/kafka-logs-2
log.dirs = /tmp/kafka-logs

##提供给客户端响应的端口
port =6667

##消息体的最大大小，单位是字节
message.max.bytes =1000000

## broker 处理消息的最大线程数，一般情况下不需要去修改
num.network.threads =3

## broker处理磁盘IO 的线程数 ，数值应该大于你的硬盘数
num.io.threads =8

## 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
background.threads =4

## 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，算是一种自我保护机制
queued.max.requests =500

##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
host.name

## 打广告的地址，若是设置的话，会提供给producers， consumers,其他broker连接，具体如何使用还未深究
advertised.host.name

## 广告地址端口，必须不同于port中的设置
advertised.port

## socket的发送缓冲区，socket的调优参数SO_SNDBUFF
socket.send.buffer.bytes =100*1024

## socket的接受缓冲区，socket的调优参数SO_RCVBUFF
socket.receive.buffer.bytes =100*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
socket.request.max.bytes =100*1024*1024
```

```
############################################# LOG 相关 ##############################################
## 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.segment.bytes =1024*1024*1024

## 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.roll.hours =24*7

## 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.cleanup.policy = delete

## 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.minutes=7days

指定日志每隔多久检查看是否可以被删除，默认1分钟

log.cleanup.interval.mins=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没有大小限制
## 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

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

## 是否开启日志压缩
log.cleaner.enable=false

## 日志压缩运行的线程数
log.cleaner.threads =1

## 日志压缩时候处理的最大大小
log.cleaner.io.max.bytes.per.second=None

## 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.dedupe.buffer.size=500*1024*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.size=512*1024

## 日志清理中hash表的扩大因子 一般不需要修改
log.cleaner.io.buffer.load.factor =0.9

## 检查是否处罚日志清理的间隔
log.cleaner.backoff.ms =15000

## 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.min.cleanable.ratio=0.5

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

## 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.size.max.bytes =10*1024*1024

## 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.index.interval.bytes =4096

## log文件"sync"到磁盘之前累积的消息条数
```


随笔档案

阅读排行榜

评论排行榜

推荐排行榜

```
## 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.interval.messages=None

## 检查是否需要固化到硬盘的时间间隔
log.flush.scheduler.interval.ms =3000

## 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.flush.interval.ms = None

## 文件在索引中清除后保留的时间 一般不需要去修改
log.delete.delay.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
log.flush.offset.checkpoint.interval.ms =60000

------------------------------------------ TOPIC 相关 ------------------------------------------
## 是否允许自动创建topic ,若是false, 就需要通过命令创建topic
auto.create.topics.enable =true

## 一个topic ,默认分区的replication个数 ,不得大于集群中broker的个数
default.replication.factor =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
num.partitions =1

实例 --replication-factor3--partitions1--topic replicated-topic :名称replicated-topic有一个分区, 分区被复制到三个broker上。

----------------------------------复制(Leader、replicas) 相关 ----------------------------------
## partition leader与replicas之间通讯时,socket的超时时间
controller.socket.timeout.ms =30000

## partition leader与replicas数据同步时,消息的队列尺寸
controller.message.queue.size=10

## 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.time.max.ms =10000

## 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.lag.max.messages =4000

##follower与leader之间的socket超时时间
replica.socket.timeout.ms=30*1000

## leader复制时候的socket缓存大小
replica.socket.receive.buffer.bytes=64*1024

## replicas每次获取数据的最大大小
replica.fetch.max.bytes =1024*1024

## replicas同leader之间通信的最大等待时间, 失败了会重试
replica.fetch.wait.max.ms =500
```


最新评论

1. Re:kafka 配置文件参数详解
mark
--byebai
2. Re:如何去除有道云笔记广告（windows）
多谢!
--爱吃猫的鱼儿
3. Re:IDEA修改选取单词颜色和搜索结果的颜色
感谢!!!
--Bigzhu
4. Re:Mycat问题总结
请问 LZ有没有遇到过 节点扩容后时间类型的字段都少了8小时的问题 我看了下中间文件 在生成插入语句的是 时间已经不对了 用的是自带的dataMigrate脚本
--你有小苹果吗
5. Re:Mycat问题总结
@H2deNote
方法事务需要加上readonly=false
--雷震子er

```
## 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
replica.fetch.min.bytes =1

## leader 进行复制的线程数, 增大这个数值会增加follower的IO
num.replica.fetchers=1

## 每个replica检查是否将最高水位进行固化的频率
replica.high.watermark.checkpoint.interval.ms =5000

## 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.enable =false

## 控制器关闭的尝试次数
controlled.shutdown.max.retries =3

## 每次关闭尝试的时间间隔
controlled.shutdown.retry.backoff.ms =5000

## 是否自动平衡broker之间的分配策略
auto.leader.rebalance.enable =false

## 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.per.broker.percentage =10

## 检查leader是否不平衡的时间间隔
leader.imbalance.check.interval.seconds =300

## 客户端保留offset信息的最大空间大小
offset.metadata.max.bytes

---------------------------------ZooKeeper 相关---------------------------------
##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.connect = localhost:2181

## 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.session.timeout.ms=6000

## ZooKeeper的连接超时时间
zookeeper.connection.timeout.ms =6000

## ZooKeeper集群中leader和follower之间的同步实际那
zookeeper.sync.time.ms =2000
配置的修改
其中一部分配置是可以被每个topic自身的配置所代替, 例如
新增配置
bin/kafka-topics.sh --zookeeper localhost:2181--create --topic my-topic --partitions1--replication-factor1--config max.message.bytes=64000--config flush.messages=1

修改配置
bin/kafka-topics.sh --zookeeper localhost:2181--alter --topic my-topic --config max.message.bytes=128000

删除配置 :
bin/kafka-topics.sh --zookeeper localhost:2181--alter --topic my-topic --deleteConfig max.message.bytes
```

二 CONSUMER 配置

最为核心的配置是group.id、zookeeper.connect

```
## Consumer归属的组ID, broker是根据group.id来判断是队列模式还是发布订阅模式, 非常重要
 group.id

## 消费者的ID, 若是没有设置的话, 会自增
 consumer.id

## 一个用于跟踪调查的ID , 最好同group.id相同
```

```
 client.id = group id value

## 对于zookeeper集群的指定，可以是多个 hostname1:port1,hostname2:port2,hostname3:port3 必须和broker使用同样的zk配置
 zookeeper.connect=localhost:2182

## zookeeper的心跳超时时间，查过这个时间就认为是dead消费者
 zookeeper.session.timeout.ms =6000

## zookeeper的等待连接时间
 zookeeper.connection.timeout.ms =6000

## zookeeper的follower同leader的同步时间
 zookeeper.sync.time.ms =2000

## 当zookeeper中没有初始的offset时候的处理方式 。smallest ：重置为最小值 largest:重置为最大值 anythingelse:抛出异常
 auto.offset.reset = largest

## socket的超时时间，实际的超时时间是:max.fetch.wait + socket.timeout.ms.
 socket.timeout.ms=30*1000

## socket的接受缓存空间大小
 socket.receive.buffer.bytes=64*1024

##从每个分区获取的消息大小限制
 fetch.message.max.bytes =1024*1024

## 是否在消费消息后将offset同步到zookeeper，当Consumer失败后就能从zookeeper获取最新的offset
 auto.commit.enable =true

## 自动提交的时间间隔
 auto.commit.interval.ms =60*1000

## 用来处理消费消息的块，每个块可以等同于fetch.message.max.bytes中数值
 queued.max.message.chunks =10

## 当有新的consumer加入到group时，将会reblance，此后将会有partitions的消费端迁移到新
## 的consumer上，如果一个consumer获得了某个partition的消费权限，那么它将会向zk注册
##"Partition Owner registry"节点信息，但是有可能此时旧的consumer尚没有释放此节点，
## 此值用于控制，注册节点的重试次数.
 rebalance.max.retries =4

## 每次再平衡的时间间隔
 rebalance.backoff.ms =2000

## 每次重新选举leader的时间
 refresh.leader.backoff.ms

## server发送到消费端的最小数据，若是不满足这个数值则会等待，知道满足数值要求
 fetch.min.bytes =1

## 若是不满足最小大小(fetch.min.bytes)的话，等待消费端请求的最长等待时间
 fetch.wait.max.ms =100

## 指定时间内没有消息到达就抛出异常，一般不需要改
 consumer.timeout.ms = -1
```

三 PRODUCER 的配置

比较核心的配置：metadata.broker.list、request.required.acks、producer.type、serializer.class

```
## 消费者获取消息元信息(topics, partitions and replicas)的地址，配置格式是:host1:port1,host2:port2，也可以在外面设置一个vip
 metadata.broker.list
```

```
##消息的确认模式
 ##0:不保证消息的到达确认，只管发送，低延迟但是会出现消息的丢失，在某个server失败的情况下，有点像TCP
 ##1:发送消息，并会等待leader 收到确认后，一定的可靠性
 ## -1:发送消息，等待leader收到确认，并进行复制操作后，才返回，最高的可靠性
 request.required.acks =0

## 消息发送的最长等待时间
 request.timeout.ms =10000

## socket的缓存大小
 send.buffer.bytes=100*1024

## key的序列化方式，若是没有设置，同serializer.class
 key.serializer.class

## 分区的策略，默认是取模
 partitioner.class=kafka.producer.DefaultPartitioner

## 消息的压缩模式，默认是none，可以有gzip和snappy
 compression.codec = none

## 可以针对默写特定的topic进行压缩
 compressed.topics=null

## 消息发送失败后的重试次数
 message.send.max.retries =3

## 每次失败后的间隔时间
 retry.backoff.ms =100

## 生产者定时更新topic元信息的时间间隔 ，若是设置为0，那么会在每个消息发送后都去更新数据
 topic.metadata.refresh.interval.ms =600*1000

## 用户随意指定，但是不能重复，主要用于跟踪记录消息
 client.id=""

----------------------------------------- 消息模式 相关 -----------------------------------------
 ## 生产者的类型 async:异步执行消息的发送 sync:同步执行消息的发送
 producer.type=sync

## 异步模式下，那么就会在设置的时间缓存消息，并一次性发送
 queue.buffering.max.ms =5000

## 异步的模式下 最长等待的消息数
 queue.buffering.max.messages =10000

## 异步模式下，进入队列的等待时间 若是设置为0，那么要么进入队列，要么直接抛弃
 queue.enqueue.timeout.ms = -1

## 异步模式下，每次发送的最大消息数，前提是触发了queue.buffering.max.messages或是queue.buffering.max.ms的限制
 batch.num.messages=200

## 消息体的系列化处理类 ，转化为字节流进行传输
 serializer.class= kafka.serializer.DefaultEncoder
```

转载：原文地址 https://blog.csdn.net/wackycrazy/article/details/47810741

分类: Java


关注 - 4
粉丝 - 7
+加关注

« 上一篇： Hyperledger Fabric 1.0 学习搭建 （二）--- 源码及镜像文件处理
» 下一篇： IntelliJ IDEA下"Cannot resolve symbol 'log'"的解决方法

posted @ 2018-03-26 15:35 小小小小小小鱼 阅读(47238) 评论(1) 编辑 收藏 举报