## Kafka部分

1. kafka集群扩容和数据迁移？

集群扩容：

利用kafka-ressign-partitions.sh工具

①新启动brokers

注意brokerid以及/etc/hosts的添加和修改，此时仅仅是增加了broker，但是topic数据并不会均分到新增的节点上

②生成分配计划generage

./bin/kafka-reassign-partitions.sh --zookeeper 192.168.2.225:2183 --topics-to-move-json-file migration-push-token-topic.json --broker-list "101,,102,103,104,105,106" --generate

关于脚本内容网上有，这一步的目的是把原来101 102 103上的指定的topics数据均分到扩容后的集群。

③执行分配execute

根据第一步生成的分配计划

kafka-reassign-partitions.sh --zookeeper $ZK\_CONNECT --reassignment-json-file topic-reassignment.json --execute

④检查分配状态verify

kafka-reassign-partitions.sh --zookeeper $ZK\_CONNECT --reassignment-json-file topic-reassignment.json --verify

数据迁移分两种情况：

第一：是否是均分的数据迁移，例如123 -> 23456

第二：是否是单一broker数据迁移，比如更换某一台broker例如 123 -> 234

第一种情况解决方案：完全可以按照上述集群扩容的方式做，只需要改动将101,102,103,104,105,106中去掉不再使用的broker即可，比如102,103,104,105,106，去掉了某一台服务器，剩下的数据均分到5台上，当然也可以不去除服务器仅仅是将某些topic方到456上等等各种情况都属于第一种

第二种情况解决方案：不需要generate、execute、verify三部，只需要后两步，只需要写一个ressign.json文件规定将1上面所有的topic+partiion全部迁移到指定brokerid如4上即可。然后直接运行--execute。就可以完成kafka单台服务器的替换过程。

1. kafka如何保证数据高可用，副本是如何进行数据同步的，leader是如何进行选举的？

注意这个问题：如果只答出副本冗余机制的话那么面试可能会被pass哦！

①replicas副本冗余机制

②failover失败转移

其一：多副本冗余机制

说白了就是partition的replications，一个leader，多个follower

注意：任何一个partition只有leader才能进行对外的读写！！

Follower会不断的向leader拉取最新的数据，当leader有新数据的时候就进行同步。

其二：Failover失败转移

极端情况，当某个partition的leader不可用时怎么办，因为数据的读写是请求leader的啊，这种情况下肯定是要从follower中选举leader的。而这个过程是怎样的的呢

选举机制：

①普通选举

如果知识leader不可用，那么只需要从zk上的ISR列表中找一个follower作为leader即可

②脏选举机制

这种情况发生在所有的副本都不可用了，才是着分区停止服务，此时进行脏选举机制，也就是说选取任何一个激励可用follower作为leader，这种方式不能保证一致性

③禁止脏选举机制

等待ISR列表中副本复活，虽然可以保证一致性，但是需要很长时间

实现：在实际实现代码中，并没有这么简单，而是通过controller来实现，因为在实际业务中可能存在成千上万的partition，如果一台服务器宕机，可能出现很多leader不可用的情况，所以需要从ontroller，而controller是kafka随机选择的一个broker作为controller leader，作用如下：

关于controller还是再总结一下把，在下面

1. 你知道ISR吗，什么情况下会导致ISR列表中的follower见少？

先说明一点，无论是leader和follower都叫做副本！！不然创建topic的时候指定副本个数为1时难道就没有副本吗，leader也是副本

ISR——in sync replicas：指的就是数据保证最新的那些副本，保存在zk上：其中肯定有leader，还有跟leader同步的follower，为什么会有ISR，因为我们不能把保证所有的follower都是实时很快同步数据，因为他们可能处于GC或者网络等问题...无法及时同步数据，当然了正常情况下总会同步的。

所以机制是：正常情况下ISR列表中保存着所有副本，而当其中一个follower非同步时，leader就会将这个follower剔除ISR列表，等待回复同步之后再加入列表。所以后面介绍的acks参数针对的是ISR列表而不是真正所有的follower，不然的话延时就大了！

从ISR剔除follower的情况：

①增加副本

②GC

③IO瓶颈

④follower失效

1. 好了问你一个重要的问题：acks参数你用过吧，你觉得它的意义何在，你顺便说说kafka同步复制和异步复制的流程吧？

首先：acks参数并不是kafka broker的参数，而是producer的参数

其次：acks参数有三个选项，分别是0,1，all

当acks=0时，表示kafkaproducer客户端只要把消息发送出去就完活，不管这条message是否落到partition的leader的磁盘上，默认发送成功

当acks=1时，这个是默认的设置！只要partition leader接受到小时而且写入本地磁盘了就认为成功了！不管follower是否同步这条message

当acks=all时，这个意思很明了了，就是leader接受到消息之后还必须要求ISR列表里跟leader保持同步的那些follwer都要把消息同步过去才能认为是成功的。

需要注意的是这三种情况都不能保证数据一定持久化/接收成功，看起来第三种acks=all是可以的，但是当副本数=1时，leader挂了根本没有follower，所以要是用acks=all必须要保证副本是>=2！

同步复制就是acks=all，异步复制就是acks=1的情况。

我们已经知道了ISR，这个列表中的副本是需要同步的partition。

同步复制：

①producer联系这块，找到leader

②想leader发送笑死

③leader收到消息并写入本地log

④follower从leader pull消息

⑤follower想本地写log

⑥follower向leader发送acks消息

⑦leader收到所有follower的ack消息

⑧leader向producer回传ack

异步复制：

.......leader写完本地log之后直接向producer恢复ack消息

1. 请问kakfa客户端缓冲机制你了解吗，kafka是如何优化这个缓冲池的？它是用来解决什么问题的？你还能列举类似的应用吗？

其实缓冲池大家并不陌生，随处可见，操作系统中更是常用，比如缓冲、缓存，socket的输入缓冲区、输出缓冲区，输入流输出流管道，sparkflink都有record的缓冲区，kafka的客户端发送数据同样也要缓冲池，namenode记录edits log时使用双缓冲机制和异步持久化机制，不是一条一条发送，而是先写入缓冲池——batch，然后一个batch一个batch的发送，那么现在就有一个问题——————GC问题，message是很多的，发送完就面临GC，那么频繁GC肯定会导致stop theworld，阻塞进程。

Kafka是如何做的呢？

其实它的做法很类似线程池、python的内存管理，当batch这块内存空间不需要的时候不会让垃圾收集器的算法标记可回收而是将其放入缓冲池中！比如给内存池32M，每个batch16kb，那么当有新数据进来时直接从内存池中拿16kb的batch即可而不用在申请内存空间了。

1. kafka是如何实现消息队列和订阅模式的？

看到这个问题，实际上也要回答一下其他消息队列如rabbitmq是如何实现的

①kafka是通过consumer group来实现的，比如消息队列的实现就是一个consumer group，同一个消费组不能重复消费一个patition中的数据，这就保证了一个消费组中所有consumer只消费一次数据，不重复；而订阅的方式实现肯定就是不同的conumer group了。

②rabbitmq则不然，它通过exchange的规则的方式将数据写入不同的queue，实现复杂路由，比如把相同数据写入多个queue，比如副歌一定规则的数据写入制定queue，这中间关键点在于routing-key和binding-key的匹配！，当然也可以直接指定queue！而consumer消费的时候并不是消费组的概念而是queue的概念，不管有多少consumer，只需要指定queue即可，如果是队列模式，那么只需要消费一个queue即可，如果是订阅模式那么多个consumer对应多个queue即可，由exchange来实现数据的copy。

另外说一下：kafka注重吞吐量，而rabbitmq注重复杂路由机制，kakfa为大数据而生，rabbitmq则是为复杂逻辑而生。

1. 既然你说有partition副本的机制保证高可用，那么你知道kafka是如何向broker上分配partitionid的吗？

注：kakfa集群中，每个broker都有均等的机会分配partition的leader机会

①将所有N歌broker和待分配i partition排序

②将第i个partition分配到i mod n的broker上

③将第i个partition的第j个副本分配到(i+j) mod n的broker上

所以partition的副本分配看上就像是一个等概论循环分配。

注意：这是理想的情况哦，也就是当topic刚刚创建时是这样分配的，但是当某台broker宕机之后这台上的leader全部都会变更，而在此恢复之后，就全部编程follower了，这样的话就不均衡了。

解决不均衡方法：

可以配置自动触发，也可以手动触发

kafka-preferred-replica-election.sh用来均衡leader

具体实现：

当最初创建topic的时候在ISR列表中的第一个副本编号就是leader，当broker宕机，leader变更之后，这个编号是不变的！！！这就是关键，所以当运行这个脚本的时候会让controller把这个编号的副本提升为leader，从而实现均衡操作。

1. 你知道kafka的持久化机制的吗？顺序写的优化你是怎么理解的，它是怎么实现的？

Kafka是基于磁盘而不是内存来存储信息的，这种方式乍一看感觉效率肯定低啊，但是实则不然，要知道顺序磁盘写的效率比随机内存写的效率要高。

①磁盘便宜，内存贵

②顺序读+预读取操作，提高缓存命中

③配合操作系统的缓存cache，这个缓存有预读取和回写计数，消费从cache读，写到cache就返回，操作系统自动flush

④内存占用太大

⑤基于文件的顺序读写

⑥持久化数据结构上选择了queue而不是btree，因为kafka只支持根据offset读取和append操作，所以基于queue的操作时O(1)，而btree时O(logn)的

⑦在大量文件读写时，基于queue的read和append只需要一次磁盘寻址，而Btree会设计多次，磁盘寻址大大降低了读写性能，因为Btree的数据存储中是根据索引确定数据存储。

1. 问了你那么多kafka的基础知识，我想问问你到底这些东西是怎么跟zk交互的？或者直接一点请问controller manager你知道吗？

Controller的目的是防止broker集群对zk的大量操作，通过这个中间件可以简化操作

功能：

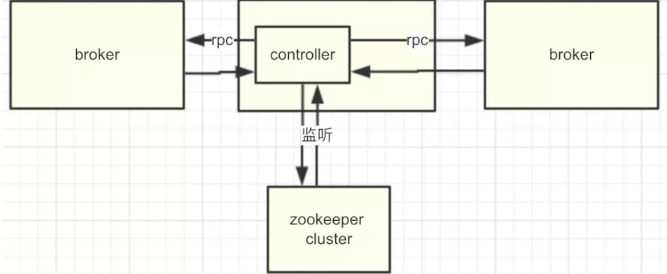
①增加删除topic

②更新分区副本

③leader选举

④集群broker增加和宕机的调整

⑤自身的选择controller leader的功能



先看一下存储在zk上的信息吧：

brokers列表：ls /brokers/ids

某个broker信息：get /brokers/ids/0

topic信息：get /brokers/topics/kafka10-topic-20170924

partition信息：get /brokers/topics/kafka10-topic-20170924/partitions/0/state

controller中心节点变更次数：get /controller\_epoch

conrtoller leader信息：get /controller

内部设计：

当controller启动时会跟集群中所有的broker建立socket连接

1. kafka的负载均衡体现在哪些方面？

①partition的分配

刚才说过了，根据分配策略，以及leader的分配和重分配

②producer的message进入分区时的负载均衡

进入策略首先：如果指定分区则进入指定分区，其次：如果没指定分区，但是有key，那么会根据key的hash进入分区，最后：如果key为空那么久轮训。

③consumer消费partition

第一种情况：如果分区大于或者等于消费者数，那么很简单无非就是一个消费者负责多个分区而已

第二种情况：如果分区数大于消费者数怎么办？默认是一对一消费，有些消费者是空闲的！！

那么如果出现多个consumer负责同一个分区会出现什么情况？就类似与多个线程操作一个对象一样，可能出现由于线程进度的问题导致消费顺序的问题，这也就是为什么kafka采用pull的方式让consumer自己拉数据而不是push，目的就是根据consumer的能力自己来消费方式压力太大或者消费顺序打乱问题。

针对第一种情况，如果分区数大于消费者，那么就有一个负载均衡的问题，到底分区怎么分配给少量的consumers？

策略一：range分配策略

简单说就是分区按数字排序，消费者按字典排序，然后基于每个topic的分区对consumer数量进行整除，不整除的话前面的消费者多分配一个分区，比如topicA的三个分区p0p1p2和topicB的p0p1p2，有两个consumer：C1和C2，那么分配则是C1：topicAp0 topicAp1 topicBp0 topicBp1，而C2：topicAp2 topicBp2，不要以为是取余！！

策略二：roundrobin轮训

这种方式就是取余的均匀轮训分配，比如上面的例子C1：topicpA0 topicAp2 topicBp1 C2:topicAp1 topicBp0 topicBp2，这种方式不局限于topic同时采用更均匀方式，当然条件是消费者们都是订阅同一个主题，如果出现订阅不相同，其实道理一样，这不过就出现不均匀的情况了。

1. 很好，最后你说一下kafka与rabbitmq的区别吧？

|  |  |  |
| --- | --- | --- |
|  | Kafka | Rabbitmq |
| 语言 | Java+scala | Erlang |
| 消息协议 | 自定义通信协议 | AMQP/MQTT/STOMP |
| 消息过滤 | Topic+partition | 交换机路由 |
| 消息堆积 | 磁盘 | 内存，也就是证明了前段时间为什么mq总蹦的问题 |
| 消息传递模式 | Pub/sub | 典型的p2p |
| 消费模式 | Pull | Push+pull |
| 消息回溯 | Offset+timestamp（文件命名） | 消费即删除 |
| 流量控制 | 对producer和consumer的设置 | Credit-based算法作用于producer |
| 消息顺序 | 支持但分区的顺序 | 单线程的发送和消费，很顺序 |
| QPS | 单机维持数十万，甚至百万 | 单机万级别 |