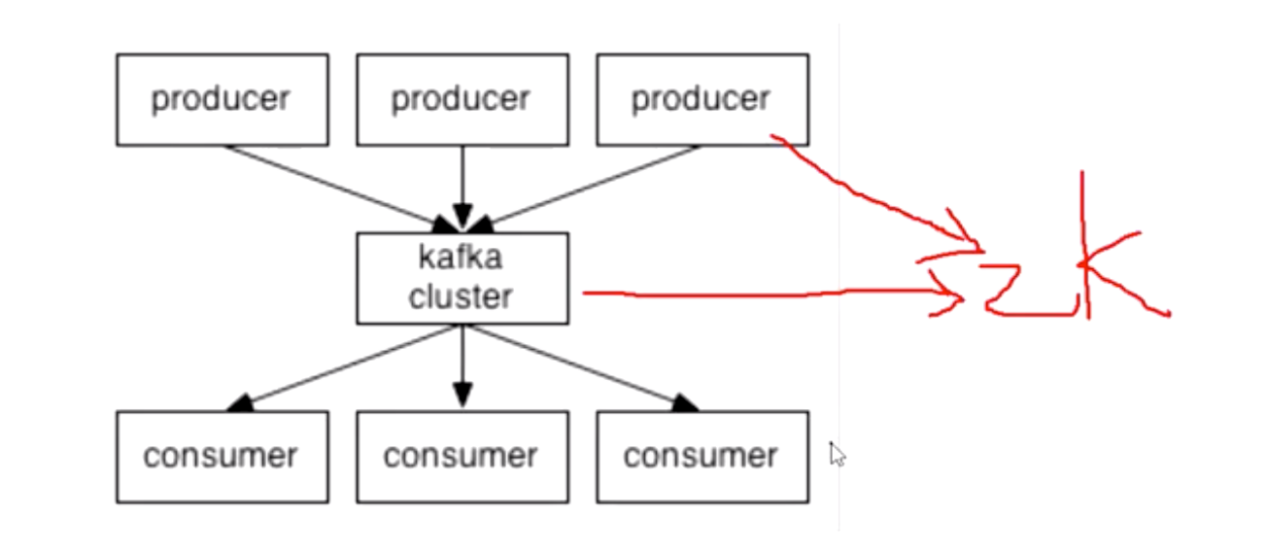
# Kafka

## Base

Replication factor备份因子.



Kafka cluster中的单个节点叫做broker.

kafka很多东西都是依赖zookeeper的.

每一个topic下面多有个partition, partition就是一个message的队列,每一个消息在队列中的序号叫做offset,对于consumer的offset是维护在zookeeper中的.

最近版的kafka呢,是会把这个offset维护在自己的一个topic \_consumer\_offset\_里的,而不是zookeeper里面,这个值的维护首先kafka会记录,然后consumer的commit操作触发offset的移动.(kafka会定期的清空consumer\_offset里面的数据,保留最新的)

有的消息队列收到consumer的正确的ack之后就会把msg删了,然后再从队列里就能拿到下一个了.而kafka不是这样的,他是为每个group都维护了一个你消费到哪里的一个offset的记录. 这个记录就是写在zookeeper里面的.

默认消息是7天消息才会被清掉.

一个topic下面设置了多个分区, 可以提高消费者的并发量,如果只有一个分区,那么同时同一个group只能有一个consumer去消费,如果你想扩大消费的并发量,你去扩几个partination就行了.

对于一个partination来说,只会有一个group中的一个consumer去消费,还有一个事就是一个group中consumer的数量最好是和partination数量一致,这样,一个consumer可以并行的消费一个partiniation,还有就是这个partination中的offset是为这个group维护的,不是单独的组中某个consumer.

一个partination只会被同一个组中的一个consumer消费,不会出现一个partinaiton被同一个group两个consumer消费了,但是一个consumer可以去消费多个partination

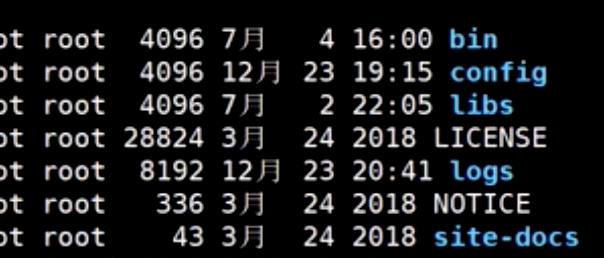
kafka能能够保证消息消费的顺序性,这个是通过个topic多个partination,然后每个consumer固定从一个partination接收消息来保证的.

如果你只有一个queue,abc三个消息,被三个consumer拿走,到底abc哪个先被执行完是没办法保证的,但是如果向kafka这样可以保证abc三个消息在所有的consumer中都是这个顺序执行的.

还有个要求就是有依赖性的消息你得都发到一个partination里面.

kafka原生就带了zookeeper.

kafka是scala语言写的,需要安装jdk.



安装目录这个德行,bin里面有很多的.sh的脚本,用来启动kafka和自带的zookeeper.

config里面有kafka和zookeeper的配置,其中的server.conf文件中有个broker.id的配置,这个配置的是钙kafka在集群中的标志id.

默认端口9092

屏幕快照 2020-03-06 下午6.20.52

这是创建topic的命令,可以看到kafka的操作人家都封好了脚本了,不是那种命令行,不同的操作还封了不同的脚本,不错.

可以看到命令中指定了用的哪个zookeeper, 多个个partination, topic的name.

屏幕快照 2020-03-06 下午6.31.17

列出所有主题,可以看到,这还有一个专门操作topic的脚本.

屏幕快照 2020-03-06 下午6.37.38

发消息的脚本 等等这种脚本.

消费消息的脚本可以指定哪个topic哪个partition的哪个offset的消息.

搭集群的时候就把配置的brokerid改了,注册到同一个zookeeper上,启动,集群自己就组建好了.

创建一个备份因子为3的topic

屏幕快照 2020-03-06 下午8.53.47

kafka的的集群也是有leader和follower的,但是不像mysql的主从,主就永远是主,从就永远是从,对不同的partination来说,leader是不一样的,也就是说,同一个,同一个topic的partination的leader都可能不同. leader是细分到partination上的,为什么这样呢,如果你一个topic下的partination都分配到同一个leader上,那么这个机器挂了,所有的partination都用不了了,都需要等重新选举之后才能用,分开的话只会影响一个.

对于一个topic,producer在发消息的时候只会发到leader,leader再同步消息到follower,类似于消息又消费了一遍.

也就是你这个consumer和这个producer,搞了这个topic,那么你会被分配到一个partination上, 然后而且你从头到位都会跟这个partination的leader producer交互,其他的follower不会有任何作用,只是备份.

kafka如果两个group消费同一个topic,那么每一个消息这两个consumer都会完整的收到,但是如果这两个consumer是属于同一个组的,那么一个消息,对于一个组来说,只会有一个consumer能收到.

我们在用代码链接kafka集群的时候,其实配置写一个kakfa实例的ip就可以,因为通过这个ipkafka可以找到zookeeper,而我们的每个topic的各个partiniation的leader信息啊,每个consumer的offset啊,都是存在zookeeper里的.

Zookeerper中保存这每个topic下的每个partition在每个group中消费的offset .其实这个offset是为一个partition的一个group维护的.

发消息的时候可以指定分区,不指定分区的话可以指定一个key,kafka会根据这个key来hash出发到哪个partination,如果key也没有的话就轮着几个partinatin去发.

发送方式有异步和同步,同步发送就是确认leader收到来,并且follower也同步到了才会算成功,异步就是调完就不管了,不过异步有回调,可以用countdownlatch来接受回调.

比如你发了五个,可以搞个5个的countdownlatch,在kafka回调函数里面去countdown 1.

在最外面去await.

这样就可以用countdownlatch把回调串起来了.

producer的一些配置

Acks 0 直接发,不会等broker操作完. 1 代表至少partination leader成功把消息写到日志文件才会给响应 -1 等待leader和所有follower都同步完才可以 默认是1 金融项目要设置成-1

前两种都是会丢消息的,第一个leader没写完挂了,follower其实还没消息,第二个leader写完了,但是还没来的及同步完,leader挂了也不行.

还可以配置重试次数和间隔,如果producer发了消息后长时间收到到broker的响应就会触发重试.

Buffmemery 消息缓冲,发消息的时候先缓冲到producer的buff中,然后慢慢的往broker去发,默认32m,还有个参数batch,就是一次性从buff里面取多少消息发到broker.还有个叫lingers,就是过了多久这个batch还没满,就不等了直接发出去.

consumer配置,如果我们不指定partination的话,只指定了topic的话,那么这个consumer会去所有的partination轮着消费的,所以解决了我们的问题.当然也可以指定分区消费.

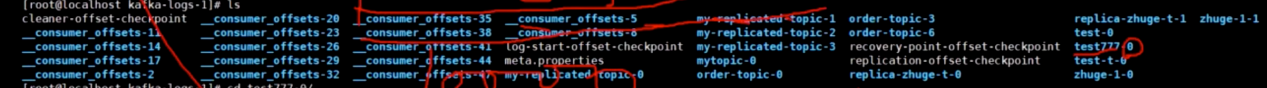
正常的话consumer会根据kafka给自己维护的offset去消费,consumer也可以自己指定从哪个offset开始消费.

还可以指定到某个时间点开始消费,比如我分析一个用户今天的行为日志,发现很异常,那么可能是我的分析代码错误了,我需要重新消费这个用户的今天的kafka消息,那么就可以指定时间点,然后kafka会告诉这个我这个时间点各个partiniatinn的offset,那么我们就可以再使用指定offset再去消费了.

consumer提交方式,首先可以手动commit.你也可以设置autocommit为true,这样kafka会比如处理完1000条消息自动去commit.自动提交的好处就是效率高,不用每次处理完都去commit一次,坏处就是可能会有重复消费,比如你消费了500条,还没有提交,准备消费一千条再统一提交,把offset直接加1000,这时候你consumer死机重启了,你去broker获取你消费的offset,获取到的是之前的,500条消费过的还是会再消费一遍.

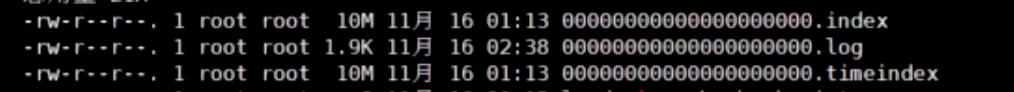
当consumer commit一个消息 kafka的consumer\_offset也会收到一个消息来维护offset的移动.

你想,这个consumeroffset这个topic的压力会比较大,每个consumergroup commit一次都会给他也提交个消息,这个topic是有50个分区的.



这个是kafka的data目录,可以看到一个partination有一个自己的文件夹,partination的名字就是topicname-01234 partiaiton的id.

也就是文件夹的单位是partination,同一个topic的partination名字开头相同.



点进一个partination的文件夹瞅瞅,

首先可以看到一个topic由于分了不同的partiniation,也就有多个日志文件,这样并发量会大,因为不是操作同一个文件,而是多个文件.

其中的.log文件就是真的消息内容.

.index是根据offset找到.log中消息的位置的索引

.timeindex是根据时间找到对应时间的消息在.log文件中位置的索引.

consumer和broker之间是长链接,长链接就是那种心跳的,你可以配置心跳时间的.

consumer拉消息也是一批一批拉的.

多个broker中会有一个被选为总控制器,controller.

controller负责的事情很多

1,分区的leader故障了,选新的leader

2,某个分区的元数据发生变化时通知其他分区更新信息

3,当给topic增加分区时,controller负责分区重新分配

4,增加broker的时候,也由controller去操作数据同步

等

controller第一次选举就是所有的broker启动的时候走去zk里面创建一个临时节点,谁成功了谁就是controller.

如果这个contoller节点挂了,首先和zk的链接也断了那么临时节点就没了,其他的broker都会监听这个临时节点,当其他的broker监听到这个节点消失了,就会触发creatre controller node, 谁成功了谁就是新的controller.

Isr 该partination leader已同步的follower节点.

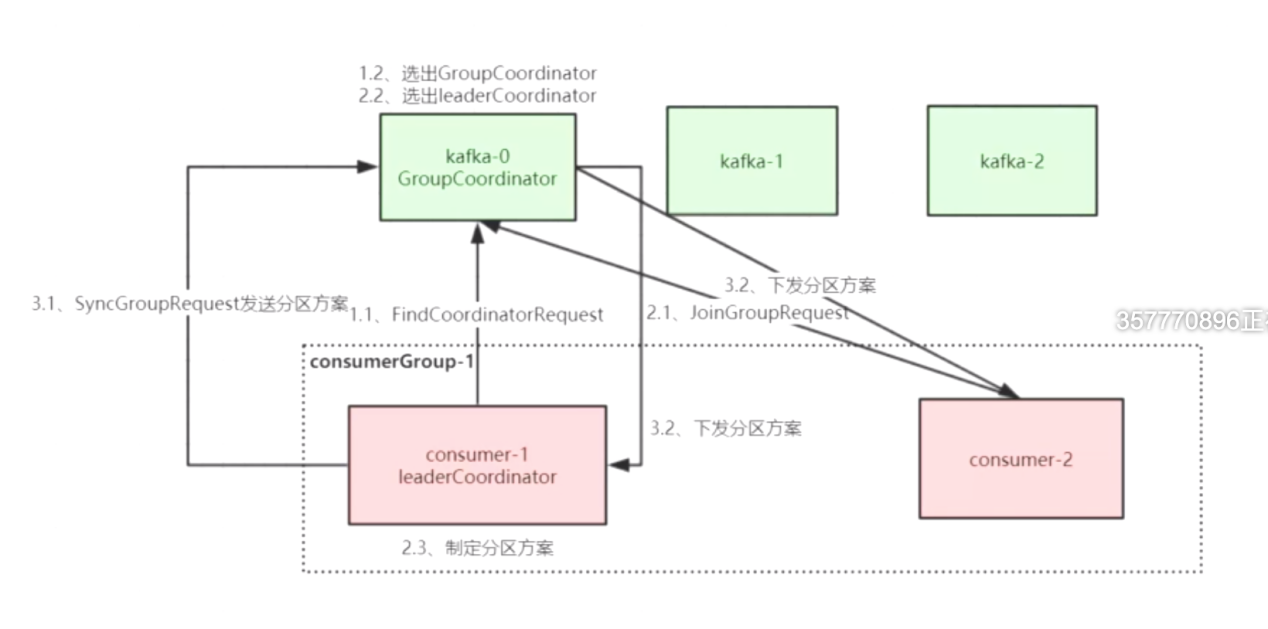
leader所在的broker挂了,全靠controller操作.

首先一个broker挂了controller是可以知道的,因为controller会监听zk中的一个东西佳偶走broker/ids 这里面有多brokerid,如果一个人broker挂了,ids就会变会被controller监听到,然后controller会根据zk里面存的元数据可以知道这个挂了的broker是那些partination的leader,这些partinaiton都需要重新选举,选举就是选择第一个在isr里面的节点作为leader.

组协调器groupcoordinator和leadercoordinator

每个group都会选择一个broker作为自己的组协调器,负责监控group里面的每个consumre的心跳,判断是否死机,如果死机了组协调器就会触发一个rebalance.

选组协调器也有个机制.



上图流程按照序号是这样的,首先这个group第一次链接broker,会发起一个finfcoordinator的请求,broker会选出一个作为这个group的组协调器,组协调器就开始工作了.组协调器还会选择一个consumer作为这个group的leadercoordinator, 这个leadercoordinator的作用就是指定分区方案,就是哪个consumer消费哪个或者哪几个partination.并把方案回传给groupcoordinator.这个group新加了别的consumer,新consumer会发送jsongroupreq请求,然后重新制定分区方案.

反正consuemr之间是互不相通的,分区方案的通知是借助于groupcoordinator同步的.

Rebalance 发生时，Group 下所有 consumer 实例都会协调在一起共同参与，kafka 能够保证尽量达到最公平的分配。但是 Rebalance 过程对 consumer group 会造成比较严重的影响。在 Rebalance 的过程中 consumer group 下的所有消费者实例都会停止工作，等待 Rebalance 过程完成。

组成员个数发生变化。例如有新的 consumer 实例加入该消费组或者离开组。

订阅的 Topic 个数发生变化。

订阅 Topic 的分区数发生变化。

看到了么, 如果你一个group订阅了多个topic,其中一个topic的分区数发送变化了,触发了rebalance,整个group 都会有影响.

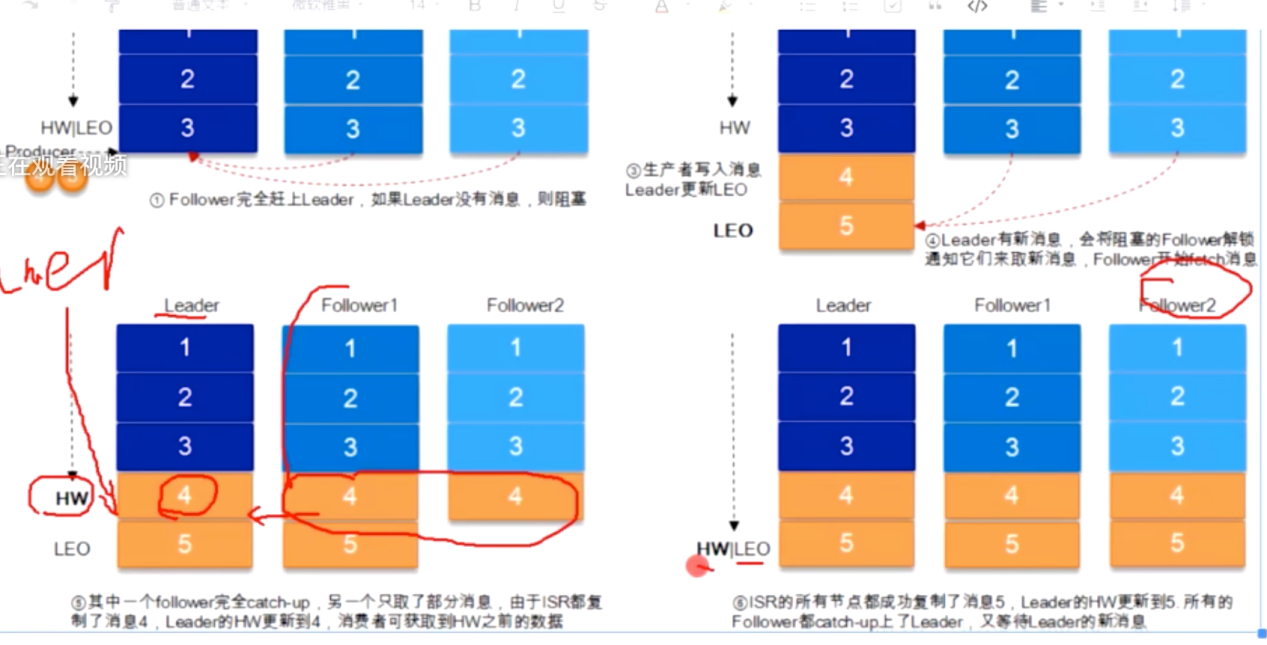
一般都是实例数量变化才会触发rebanlance,触发后,所有的consuemr都会发送syn group请求,然后组协调器选出,leadercoordinator,然后leader指定好分区方案,告诉groupcoordinator,然后再同步到所有组.

分配规则也就是那样咯 无非是012 345 678

要么就036 147 258这样了

leader和follower之间同步消息是,leader为每一个follower都开一个线程,消息同步是follower主动去批量的pull message的.

Hw 高水位 和leo的概念

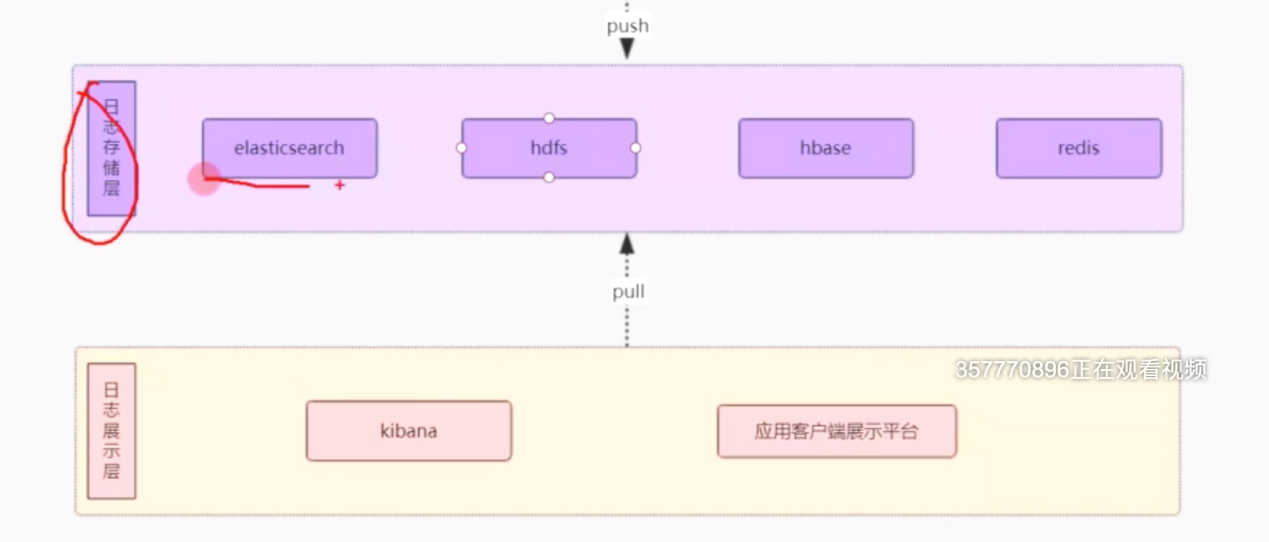
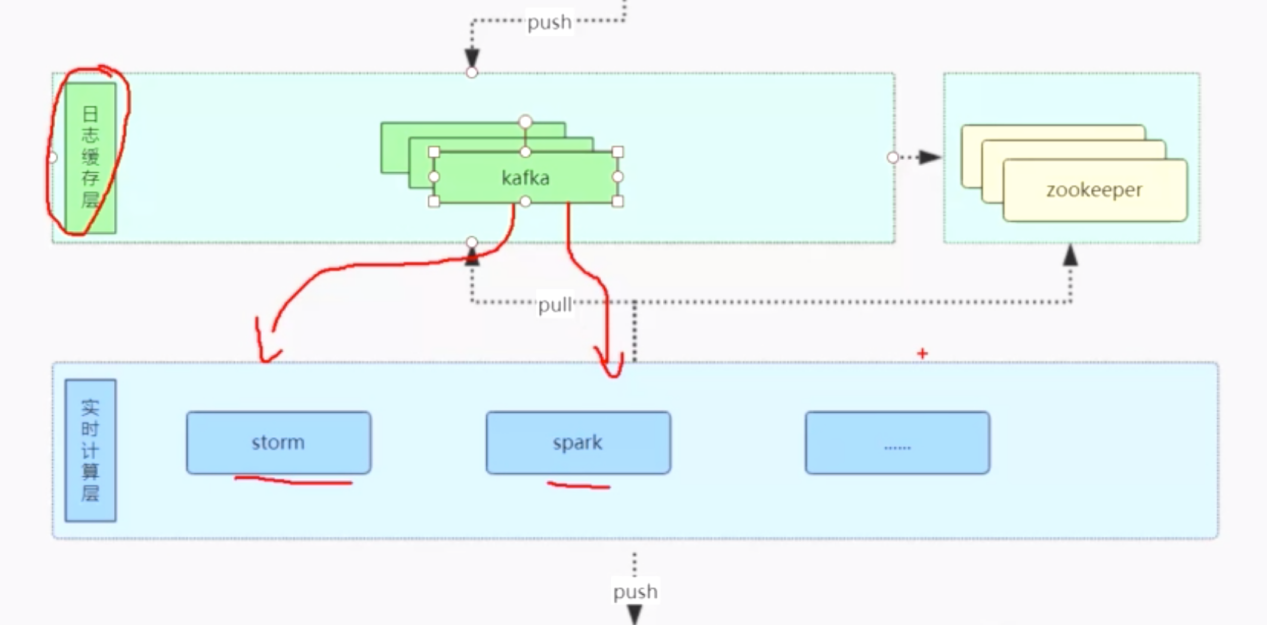
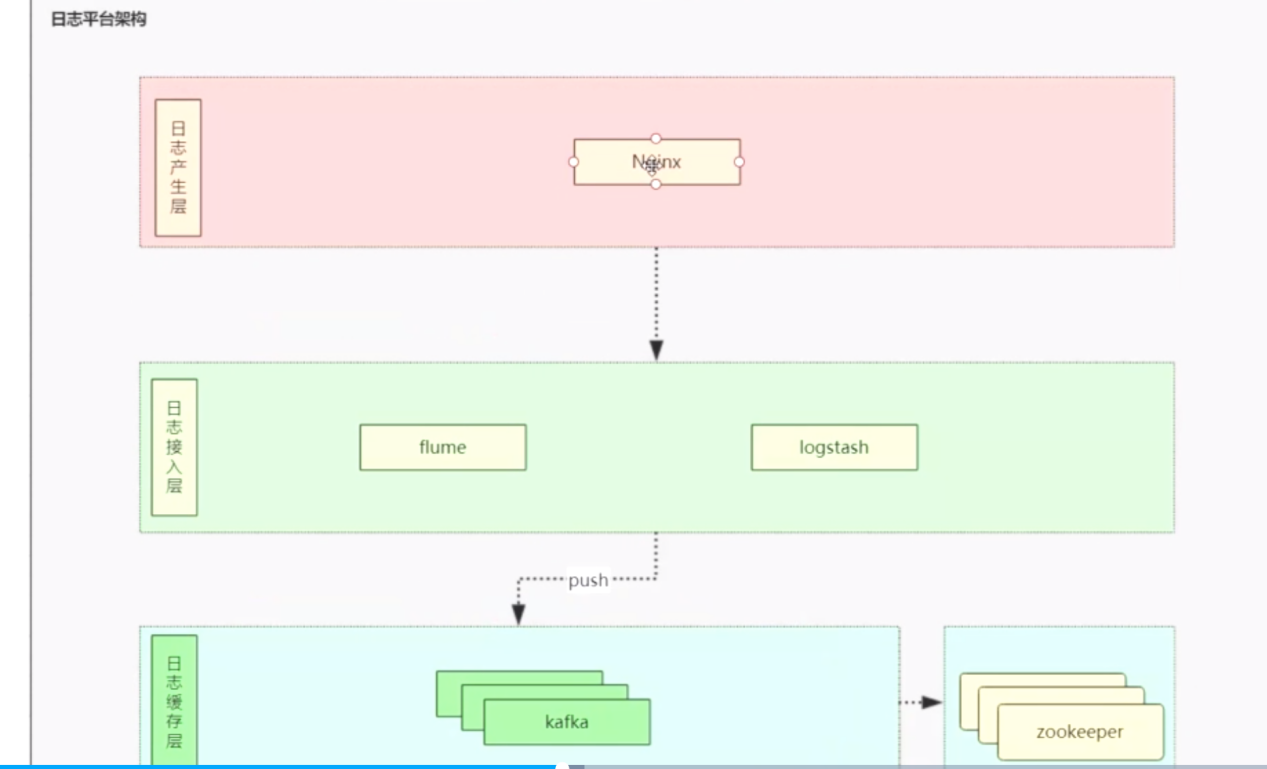


hw就是可以被消费到的消息的index.leo就是最大的offset.

hw就是leader和follower都同步完的index.

Kafka manager一个可视化的管理平台,可以监控所有的概念的东西.

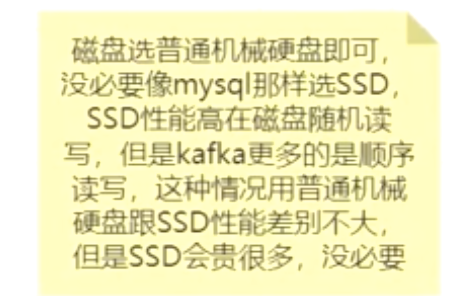
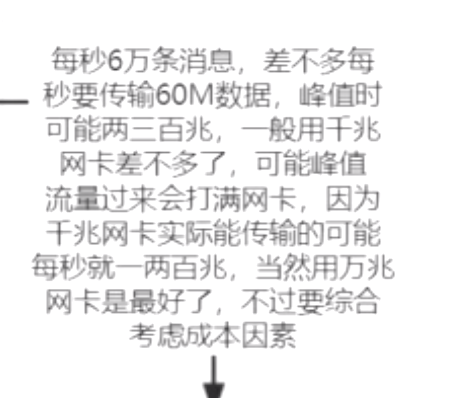
日志系统架构



kafka一个broker在普通的物理机器上每秒能处理10w消息.

一般topic的副本数也就是follower+leader也就是2.

kafka默认的消息保存也就是7天.



mysql的服务器都要用ssd,kafka用机械硬盘就可以了.

kafka为什么这么快,主要是因为他的磁盘操作都是顺序读,顺序写,人家本来就是队列么.

kafka肯定是要大内存的,消息都是要在kafka内存中中转到磁盘的.

## 问题

丢消息场景

Acks=0,这种情况,producer发完就不管了,消息有没有存到log,有没有同步到broker都不管.网络抖动, producer宕机就会丢消息

重复消息

producer网络抖动,没有收到producer的ack就会导致重复消费,producer发了多次

这里的意思就是如果kafka长时间没有收到你的响应,是会自动再次发送的,而消息消费失败,这个要看consumer的处理方式了,如果你还是自动提交了,那么你就把offset更新了,就跳过了,如果你返回给kafka失败,那么就要看kafka有没有配置重试策略了和重试次数了.

还有就是自动提交的时候消费了500个, consumer宕机了,offset没有更新,这个500等consuemr启动后还会去消费.

消息乱序

同一个partination内顺序理论上是可以保证的,但是呢,如果你想发123, 首先我们知道发送消息的时候是有缓存的,是分批发送的,如果这个123都是在一批还好,万一不再一批,1这批发完了,23批发送, 但是1这批的1失败了,导致1重试了,重试的1就到后面去了.

但是为啥1那边抖动了,23还让发呢,1那批有失败的后面的就不应该发了,kafka是有个容忍度的,如果前面的批次失败数量小于5个,还是让发的.

对于上面这种我们就不要用重试机制了,或者你就同步发送好了,不要异步分批次发送了,根据你的场景来看.

消息堆积

比如producer很多,发的都很快,就积压了,注意kafka加快消费速度不是简单的扩consuemr,因为一个partination只能被一个group中的consumer去消费,所以如果你partination只有2 ,你consuemr弄到20速度也不会加快,这种情况不仅要扩展consumer,也要扩展partination.

如果扩展来不及了,你可以改consumer的情况,让这个consuemr转发到别的topic,别的topic做好扩展.

还有就是比如的代码bug了,consumer消费总是失败,你可以搞一个死信队列,把这种处理不了的消息转发到死信队列,避免当前topic一直重试导致堆积.

用kafka实现延时队列.

消息回溯 就是觉得代码有bug,把某天的消息重新消费一遍.

kafka的bin目录是自带压测脚本的,你可以试试,并不是partination越高,吞吐量就越高,加到一个临界值再加就会降低了,因为磁盘io,内存,网络啥的都会制约你.

## Tips

Ssd贵就贵在随机读写的.

千兆网卡理论最大传输速度是(1024Mbit/s=128MByte/s).

不要用一个consumer消费多个topic,为啥,因为如果一个topic的消费出问题了,也会影响另一个.

kafka是consumer去pull的模型,这才是比较靠谱的.

如果需要实现广播，只要每个Consumer有一个独立的Group就可以了。要实现单播只要所有的Consumer在同一个Group里。用Consumer Group还可以将Consumer进行自由的分组而不需要多次发送消息到不同的Topic。

什么叫客户端代码,就是我们的程序代码为了链接kafka所引用的比如java或go的kafka相关的包就是客户端代码,比如consuemr的包,producer的包.

Redis mysql同理.

对于我们的程序代码来说,mysql redis Kafka 本身的本尊是服务端,我们是程序代码是客户端,客户端代码就是用来链接中间件的包.