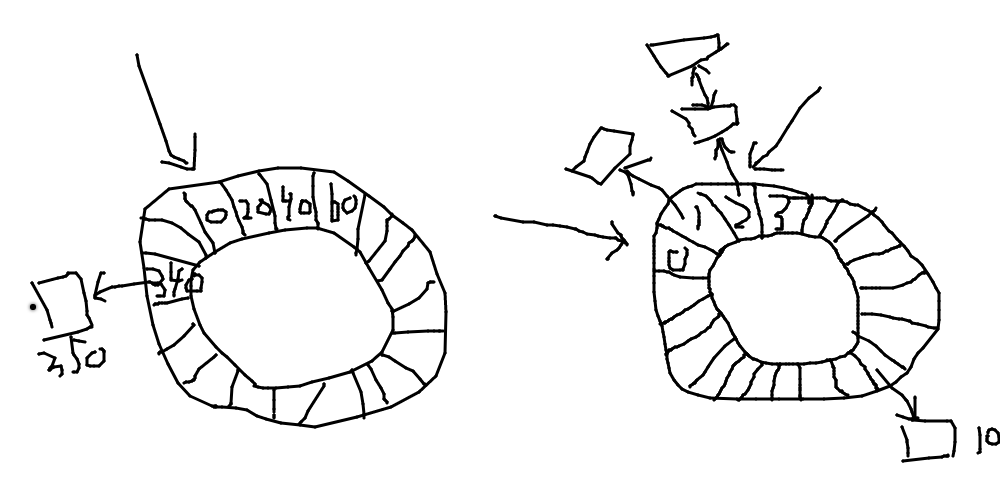
# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/125\_Kafka的时间轮延时调度机制（四）：基于DelayQueue推动/时间轮.png**



# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/147\_目前架构下可能存在的问题（13）：如何对消息流转链路进行消息轨迹监控/笔记.doc**

很多常见的一些应用，实时数据仓库，可能是从kafka里消费到数据，处理一下，写入到下一个kafka topic，再有人消费出来，再处理一下，再写入到下一个kafka topic，他是一个链路的场景

消息流转链路，面向B端的场景里，就是说你有一个链路，比如在哪个程序里消费到了一条消息，如何如何处理，写入到了下一个kafka topic去，又是哪一个程序获取到了这条消息，如何如何处理，再次写入到了下一个kafka topic

程序A -> Topic A -> 程序B -> Topic B -> 程序C -> Topic C

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/151\_下一个课程的预告：Kafka源码剖析、生产方案、性能优化以及故障处理/笔记.doc**

深入从producer端开始就剖析kafka源码，从你生产数据，broker端如何接收，写入本地，副本之间同步，LEO、HW的更新，ISR维护，时间轮延迟任务，消费数据，全链路的核心源码剖析一遍

就可以来做各种各样的生产方案了，找一些案例，来落地各种各样的方案，只是用案例做一下演示，对你出去面试是很有好处的，之前有一个同学去面试阿里，好不容易撑到了4面，阿里P9去面他

盯着kafka往深了问，分布式，高吞吐、低延迟，底层源码级别的东西，搞不定了

另外一个同学出去面试，kafka就是一直问各种生产方案，虽然暂时还是一些案例来实战，但是搞明白了以后对你出去面试很有帮助

kafka的各种生产方案，后续会在实时数据大项目里面去结合真实的项目各种落地，2C、2M、2B的各种实时数据的产品，需求和场景是不一样的，上的架构也是不一样的

kafka全链路常见的一些性能优化的方案技巧

kafka常见的异常和报错，会带着大家来看一下，如何站在源码的角度来分析和解决报错

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/135\_目前架构下可能存在的问题（01）：Kafka全链路数据丢失风险分析/笔记.doc**

万一在producer缓冲区的时候，直接就生产服务就挂了，此时你写出去的数据不就丢失了吗？

acks = 1，那么只要leader写入成功了，就可以认为是成功，但是如果数据刚刚写入leader成功，客户端也认为成功了，但是follower还没同步数据，此时leader宕机了，必然会导致数据的丢失

acks = -1，leader + follower，leader和follower都写成功了，才算是成功，此时任何一个副本宕机，都不会导致数据的丢失；如果min.insync.replicas = 1，此时如果follower先宕机了，导致ISR里就一个leader了

此时acks = -1，ISR里的副本都写入成功，leader写入成功，就算成功了，结果leader也挂了，此时数据还是会丢失的

如果你消费到数据，poll到了数据，还没来得及处理，人家就自动提交offset了，此时kafka就认为你已经成功处理了这批数据，但是此时你consumer直接宕机了，数据丢失了，但是offset已经提交了

就是按照这个kafka课程，你如果上了生产，其实会经常发现丢数据的情况的，订单系统支付成功以后就会发送订单到kafka，会员系统需要来增加积分，如果丢失一些订单，会导致很多人支付以后，会员积分没有累加

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/126\_目前为止对Kafka架构原理学习到的知识体系总结（一）：Broker端/笔记.doc**

kafka全链路架构，broker端、生产端、消费端

生产集群规划和部署，监控和管理，生产服务，消费服务，各个端的参数应该如何来设置，我们的一个驱动的场景，电商场景驱动，数据量，10亿消息量的场景下，要存储多大量的数据，要支撑多高的并发

broker端机器数量和配置，生产端的机器数量和配置，消费端的机器数量和配置

（1）集群架构：Controller -> zk -> broker

（2）元数据管理：Controller，Topic -> Partition -> Leader & Follower -> Broker

（3）分布式架构：Topic逻辑数据集合 -> 物理上的多个Partition -> 分布在多台Broker

（4）broker端的多路复用模型的请求处理架构，自定义二进制协议

（5）磁盘存储，os cache，零拷贝，索引文件，日志文件，定期清理文件

（6）高可用架构：多副本冗余机制，Leader & Follower角色分配，leader -> follower的副本同步机制，LEO & HW的更新，ISR机制，Broker宕机后如何由Controller感知以及触发Leader重新选举

（7）负载均衡架构：所有partition物理的均匀分散在broker集群，leader partition也是均匀分散在broker集群

（8）延时处理机制：时间轮实现

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/141\_目前架构下可能存在的问题（07）：如何实现延迟队列的效果/笔记.doc**

举这么一个场景，你在生产消息的时候，如果说发现某个队列现在压力很大，或者是下游消费者压力很大，你是不是可以不要继续把消息拼命的往下游发送去执行，你可以不可以让消息进入特殊的延迟队列

你进入kafka之后，我指定你延迟30秒后才能被消费到，DelayQueue

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/150\_盘点一下学员遇到的各种Kafka生产环境莫名报错都有哪些场景/笔记.doc**

细粒度的调优，保证了吞吐量和延迟都在把控的范围内，解决可能遇到的一些报错和异常

kafka的异常并没有那么的多

（1）kafka升级重启，leader找不到

（2）coordinator，提交了报错，consumer频繁报错，提交offset失败，找不到对应的coordinator，这个报错就是很多，比如说是不是网络问题，导致你跟broker之间的通信有异常，连接不到coordinator

（3）ISR，follower突然之间不从leader去同步副本了，要不然是follower的broker在fullgc卡住了，要不然可能也是两台机器之间的网络故障

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/136\_目前架构下可能存在的问题（02）：为什么线上数据总是莫名其妙出现重复/笔记.doc**

数据重复这个问题其实也是挺正常，全链路都有可能会导致数据重复

生产端是有重试机制，发现你有一些网络抖动，在底层的网络环节，其实消息发送出去了，结果收到了一个NetworkException，此时会重发消息

消费端，重复的问题，很频繁，如果你是用自动offset提交，一定会每次重启consumer服务的时候，一定会重复消费消息，刚刚接收到一批消息，poll出来的，处理完毕了，自动提交offset还没来得及执行，你就重启了

会从上一次提交offset的地方重新拉取消息再次执行一遍，消息重复

kafka，同步订单，结果每天都发现有重复的订单同步的现象，支付了一个订单，应该累加积分是1000，结果订单重复同步了一次，导致人家积分增加了2000

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/134\_基于对Kafka架构原理的理解合理设置各种参数（三）：消费服务参数/笔记.doc**

broker端的架构原理

（1）集群架构：Controller -> zk -> broker

（2）元数据管理：Controller，Topic -> Partition -> Leader & Follower -> Broker

（3）分布式架构：Topic逻辑数据集合 -> 物理上的多个Partition -> 分布在多台Broker

（4）broker端的多路复用模型的请求处理架构，自定义二进制协议

（5）磁盘存储，os cache，零拷贝，索引文件，日志文件，定期清理文件

（6）高可用架构：多副本冗余机制，Leader & Follower角色分配，leader -> follower的副本同步机制，LEO & HW的更新，ISR机制，Broker宕机后如何由Controller感知以及触发Leader重新选举

（7）负载均衡架构：所有partition物理的均匀分散在broker集群，leader partition也是均匀分散在broker集群

（8）延时处理机制：时间轮实现

生产端的架构原理

（1）元数据拉取机制：Topic -> Partition -> Leader & Follower -> Broker

（2）分区机制，默认的分区策略，hash分区策略

（3）缓冲区：数据结构，每个分区有多个batch

（4）Sender线程 + batch批量发送，按照broker来聚合多个batch作为request

（5）同步和异步

（6）核心参数：batch.size、缓冲区大小、acks、超时时间

消费端的架构原理

（1）消费模型：consumer group，每个分区就给一个consumer，queue & PUB/SUB

（2）分区分配机制：coordinator -> \_\_consumer\_offsets，JoinGroup -> leader consumer -> 下发分区方案 -> 状态机，三种分区策略

（2）消费方式：poll()调用，单线程干所有的事情

（3）offset内存数据结构更新，定时提交offset，内部topic（\_\_consumer\_offsets）

（4）故障感知机制：核心参数，心跳参数、会话超时、消费过慢

（5）老版本api的high-level和low-level，简单的认知就可以了，新版本

需求场景 -> 请求量 -> 数据量 -> 高峰并发量 -> 10亿消息

kafka集群部署：多少台物理机、磁盘规划（容量和类型）、内存规划（os cache & jvm）、CPU、网卡，10亿请求的场景，需要一个多大规模的kafka集群

kafka运维管理：支撑业务（topic创建和维护、可视化平台）、集群监控（可视化平台、开源工具）、资源使用率 -> 集群扩容 / topic扩容、机器负载率 -> 负载均衡、异常监控（报错 / jvm fullgc）、运维管理（多集群同步、安全认证、版本升级）

生产端机器部署：可能是你的java系统在生产消息，也可能是flume日志采集，canal数据库binlog采集，爬虫，如果你的机器就是每天1000万~2000万，数据往kafka里写入，没其他的数据库访问或者是别的线程和操作，4核8G的虚拟机要2台正好，高峰期的时候，CPU和内存的负载都到50%了，10亿 -> 40台4核8G虚拟机 -> 10台~20台左右16核32G的高配置虚拟机，也可以抗住每天生产10亿条数据的负载

消费端机器部署：flink、spark streaming、自己写的java系统，4核8G虚拟机，单线程消费出来 -> 线程池来处理（60），每秒钟消费吞吐量大概可以到个几百，一两千~两三千，对应的是说，他需要对数据库进行很多增删改查的操作，16核32G，每秒消费和处理5000条数据，10多台机器，spark streaming他每秒要消费和处理几万条数据，那么需要多少的资源，cpu core、内存，集群是运行在多少台机器上的集群，spark streaming，集群规模有30，480核 + 640G，160核 + 320G

kafka集群部署好，生产者代码写好 + 部署好，消费者代码写好 + 部署好，基本上已经可以run起来了，可以抗住每秒几万的并发请求，一天10亿的数据量流过kafka在处理，对其中的一些细节进行参数设置

数据保留几天，默认是7天，min.insync.replicas配合ISR + 数据不丢失，topic默认的分区数量，分区默认的副本数量，\_\_consumer\_offsets的分区数量和副本数量，processor线程数量，Handler线程池的大小，jvm参数的设置（堆大小 & G1垃圾回收器）

生产端参数：缓冲区、batch、超时时间、acks

消费端参数：offset提交、poll拉取消息、心跳、超时、拉取过慢

其实如果你不太care一些生产问题的方案，那么其实到此为止，已经足以支撑你一套10亿消息的架构，kafka、生产服务、消费服务

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/144\_目前架构下可能存在的问题（10）：消费服务故障场景下的重试队列/笔记.doc**

下游消费者整个的就是故障了，没办法继续消费了，此时可以结合TTL，死信队列，针对一些异常故障的场景来进行处理，下游消费者是正常的，结果呢，不知道为什么，在消费以及处理某一条消息的时候，就是失败了

比如说消费服务在对MySQL操作的时候，突然出现一个dead lock，抛异常，此时就可以把这个消息放入一个特殊的队列，叫做重试队列，后面可以重新消费一遍再次处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/129\_每天10亿消息的生产项目如何基于Kafka实现（一）：Kafka集群部署/笔记.doc**

broker端的架构原理

（1）集群架构：Controller -> zk -> broker

（2）元数据管理：Controller，Topic -> Partition -> Leader & Follower -> Broker

（3）分布式架构：Topic逻辑数据集合 -> 物理上的多个Partition -> 分布在多台Broker

（4）broker端的多路复用模型的请求处理架构，自定义二进制协议

（5）磁盘存储，os cache，零拷贝，索引文件，日志文件，定期清理文件

（6）高可用架构：多副本冗余机制，Leader & Follower角色分配，leader -> follower的副本同步机制，LEO & HW的更新，ISR机制，Broker宕机后如何由Controller感知以及触发Leader重新选举

（7）负载均衡架构：所有partition物理的均匀分散在broker集群，leader partition也是均匀分散在broker集群

（8）延时处理机制：时间轮实现

生产端的架构原理

（1）元数据拉取机制：Topic -> Partition -> Leader & Follower -> Broker

（2）分区机制，默认的分区策略，hash分区策略

（3）缓冲区：数据结构，每个分区有多个batch

（4）Sender线程 + batch批量发送，按照broker来聚合多个batch作为request

（5）同步和异步

（6）核心参数：batch.size、缓冲区大小、acks、超时时间

消费端的架构原理

（1）消费模型：consumer group，每个分区就给一个consumer，queue & PUB/SUB

（2）分区分配机制：coordinator -> \_\_consumer\_offsets，JoinGroup -> leader consumer -> 下发分区方案 -> 状态机，三种分区策略

（2）消费方式：poll()调用，单线程干所有的事情

（3）offset内存数据结构更新，定时提交offset，内部topic（\_\_consumer\_offsets）

（4）故障感知机制：核心参数，心跳参数、会话超时、消费过慢

（5）老版本api的high-level和low-level，简单的认知就可以了，新版本

需求场景 -> 请求量 -> 数据量 -> 高峰并发量

kafka集群部署：多少台物理机、磁盘规划（容量和类型）、内存规划（os cache & jvm）、CPU、网卡，10亿请求的场景，需要一个多大规模的kafka集群

kafka运维管理：支撑业务（topic创建和维护、可视化平台）、集群监控（可视化平台、开源工具）、资源使用率 -> 集群扩容 / topic扩容、机器负载率 -> 负载均衡、异常监控（报错 / jvm fullgc）、运维管理（多集群同步、安全认证、版本升级）

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/130\_每天10亿消息的生产项目如何基于Kafka实现（二）：生产服务部署/笔记.doc**

broker端的架构原理

（1）集群架构：Controller -> zk -> broker

（2）元数据管理：Controller，Topic -> Partition -> Leader & Follower -> Broker

（3）分布式架构：Topic逻辑数据集合 -> 物理上的多个Partition -> 分布在多台Broker

（4）broker端的多路复用模型的请求处理架构，自定义二进制协议

（5）磁盘存储，os cache，零拷贝，索引文件，日志文件，定期清理文件

（6）高可用架构：多副本冗余机制，Leader & Follower角色分配，leader -> follower的副本同步机制，LEO & HW的更新，ISR机制，Broker宕机后如何由Controller感知以及触发Leader重新选举

（7）负载均衡架构：所有partition物理的均匀分散在broker集群，leader partition也是均匀分散在broker集群

（8）延时处理机制：时间轮实现

生产端的架构原理

（1）元数据拉取机制：Topic -> Partition -> Leader & Follower -> Broker

（2）分区机制，默认的分区策略，hash分区策略

（3）缓冲区：数据结构，每个分区有多个batch

（4）Sender线程 + batch批量发送，按照broker来聚合多个batch作为request

（5）同步和异步

（6）核心参数：batch.size、缓冲区大小、acks、超时时间

消费端的架构原理

（1）消费模型：consumer group，每个分区就给一个consumer，queue & PUB/SUB

（2）分区分配机制：coordinator -> \_\_consumer\_offsets，JoinGroup -> leader consumer -> 下发分区方案 -> 状态机，三种分区策略

（2）消费方式：poll()调用，单线程干所有的事情

（3）offset内存数据结构更新，定时提交offset，内部topic（\_\_consumer\_offsets）

（4）故障感知机制：核心参数，心跳参数、会话超时、消费过慢

（5）老版本api的high-level和low-level，简单的认知就可以了，新版本

需求场景 -> 请求量 -> 数据量 -> 高峰并发量 -> 10亿消息

kafka集群部署：多少台物理机、磁盘规划（容量和类型）、内存规划（os cache & jvm）、CPU、网卡，10亿请求的场景，需要一个多大规模的kafka集群

kafka运维管理：支撑业务（topic创建和维护、可视化平台）、集群监控（可视化平台、开源工具）、资源使用率 -> 集群扩容 / topic扩容、机器负载率 -> 负载均衡、异常监控（报错 / jvm fullgc）、运维管理（多集群同步、安全认证、版本升级）

生产端机器部署：可能是你的java系统在生产消息，也可能是flume日志采集，canal数据库binlog采集，爬虫，如果你的机器就是每天1000万~2000万，数据往kafka里写入，没其他的数据库访问或者是别的线程和操作，4核8G的虚拟机要2台正好，高峰期的时候，CPU和内存的负载都到50%了，10亿 -> 40台4核8G虚拟机 -> 10台~20台左右16核32G的高配置虚拟机，也可以抗住每天生产10亿条数据的负载

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/149\_如何对线上Kafka全链路进行高吞吐、低延迟的性能优化/笔记.doc**

你首先在线上需要针对一些问题设计对应的生产方案，解决各种问题

随之而来的就是做一些细节性的调优，消息流转质量的监控，平均每条消息流转完一个链路的延迟，消息从第一次发送到kafka里去，到最终计算完毕写入存储中，一共耗时多长时间，几百毫秒

在链路流转的过程中，每个环节，可能spark streaming，有很多个作业都会对一个消息进行处理，java服务对一个消息进行处理，消息流转经过每个环节耗时多久

kafka调优：低延迟、高吞吐

如果你一个消息进入kafka到流转完一条链路出来，需要耗时几分钟，或者是几十分钟的话，那么就一定是有问题的

kafka延迟的地方，主要是几块，生产端，batch.size过大，linger.ms过大，会导致消息的延迟就比较大，acks对延迟的影响是很大的

broker端，主要是processor、handler线程数量要足够多才能高效率高吞吐的处理各种消息的进入，os cache一定要关注好，acks是有关系的

consumer端，主要是你拉取到一条消息后交给线程池来处理的，如果你的线程池处理的太慢，或者是数量不够多，线程池都满了，此时拉取到了消息也没办法来处理，此时会导致吞吐量下降，延迟提高

高吞吐，也是通过消息的监控来观察的，你需要去看一下最开始写入kafka的吞吐量是每秒钟多少条消息，每个环节消费消息的吞吐量，以及写入下一个topic的吞吐量，每秒钟分别是多少条

观察出来整个链路里，各个环节他的发送消息以及消费消息的吞吐量

程序A -> topic的吞吐量是每秒钟1万，topic -> 程序B的吞吐量仅仅只有每秒钟8000，那就说明程序B的吞吐量现在不高

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/138\_目前架构下可能存在的问题（04）：线上消息队列百万消息积压怎么处理/笔记.doc**

比较致命的一个问题

（1）下游的消费者，比如说他是依赖于MySQL、NoSQL之类的系统，但是现在就是说MySQL他的性能突然出现了很大的问题，比如说一个表有几千万条数据，有个同学在高峰期做了一个DDL，导致对数据库的增删改查非常的慢，性能急剧下降几十倍，瞬间导致kafka里大量的积压了很多很多的消息，几百万条消息积压在里面

（2）依赖的ES突然宕机了，ES集群突然故障了，无法访问了，机房故障，机房的冷却系统坏了，温度过高，机器都故障了，下游的消费者就不工作了，积压几百万的消息

（3）你怎么来处理，应对

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/139\_目前架构下可能存在的问题（05）：如何配合分布式事务方式实现消息事务支持/笔记.doc**

无论是Java还是大数据的同学，都需要学习分布式系统的环节，项目阶段二，spring cloud + 分布式事务 + 分布式锁 + 幂等性，这套东西组成了一个最最基本的分布式业务系统的架构，大数据的同学也得了解一下

可靠消息最终一致性的事务方案，基于MQ来实现的，可靠消息服务，上游服务的本地事务成功了，必须保证消息投递出去到MQ上去，可以由MQ自己来支持和实现，RocketMQ就实现了一套机制

提供了事务相关的支持，数据库事务 + 消息，封装在一个事务里，数据库事务成功了，消息也必须投递成功，如果要跟分布式事务来整合的话，一般用的不是Kafka，RocketMQ来支持就最好

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/127\_目前为止对Kafka架构原理学习到的知识体系总结（二）：生产端/笔记.doc**

broker端的架构原理

（1）集群架构：Controller -> zk -> broker

（2）元数据管理：Controller，Topic -> Partition -> Leader & Follower -> Broker

（3）分布式架构：Topic逻辑数据集合 -> 物理上的多个Partition -> 分布在多台Broker

（4）broker端的多路复用模型的请求处理架构，自定义二进制协议

（5）磁盘存储，os cache，零拷贝，索引文件，日志文件，定期清理文件

（6）高可用架构：多副本冗余机制，Leader & Follower角色分配，leader -> follower的副本同步机制，LEO & HW的更新，ISR机制，Broker宕机后如何由Controller感知以及触发Leader重新选举

（7）负载均衡架构：所有partition物理的均匀分散在broker集群，leader partition也是均匀分散在broker集群

（8）延时处理机制：时间轮实现

生产端的架构原理

（1）元数据拉取机制：Topic -> Partition -> Leader & Follower -> Broker

（2）分区机制，默认的分区策略，hash分区策略

（3）缓冲区：数据结构，每个分区有多个batch

（4）Sender线程 + batch批量发送，按照broker来聚合多个batch作为request

（5）同步和异步

（6）核心参数：batch.size、缓冲区大小、acks、超时时间

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/145\_目前架构下可能存在的问题（11）：下游数据计算错误时如何进行消息回溯/笔记.doc**

比如说消费者服务里的一些代码逻辑有bug，他对消息的处理是错误的，此时如果你发现这个问题之后，首先应该把错误的数据给删除掉，对那些数据，是不是要进行回溯，从kafka里重新读取某个offset~某个offset的一段数据，重新这个流程来一遍

就是重新对这些数据做一遍处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/146\_目前架构下可能存在的问题（12）：不同的业务数据如何实现消息路由/笔记.doc**

topic里，可能也细分为不同的数据，同步订单

有的订单是正常情况下支付的，就是按照正常的流程来增加积分就可以了

但是有的订单是采用的是特殊的折扣优惠的场景，说明了是无积分累加的，此时对这种特殊的订单是否可以打一个标签

特殊订单和正常订单都在一个topic里，但是下游的消费服务获取到了消息，可以进行路由，也就是仅仅路由出来正常的订单给后续的逻辑来处理，特殊订单就是不予以处理就可以了，对消息打上不同的标签，消息路由

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/142\_目前架构下可能存在的问题（08）：不同的消息如何实现多优先级队列/笔记.doc**

消息写入kafka的时候，可以指定消息的优先级，就是优先级越高的数据应该越是先被消费出来，消息可以按照优先级来排序，优先级高的先被消费出来，你可能在方那哦数据过去的时候，有些数据是大客户的数据，优先级可以指定的高一些

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/132\_基于对Kafka架构原理的理解合理设置各种参数（一）：Kafka集群参数/笔记.doc**

broker端的架构原理

（1）集群架构：Controller -> zk -> broker

（2）元数据管理：Controller，Topic -> Partition -> Leader & Follower -> Broker

（3）分布式架构：Topic逻辑数据集合 -> 物理上的多个Partition -> 分布在多台Broker

（4）broker端的多路复用模型的请求处理架构，自定义二进制协议

（5）磁盘存储，os cache，零拷贝，索引文件，日志文件，定期清理文件

（6）高可用架构：多副本冗余机制，Leader & Follower角色分配，leader -> follower的副本同步机制，LEO & HW的更新，ISR机制，Broker宕机后如何由Controller感知以及触发Leader重新选举

（7）负载均衡架构：所有partition物理的均匀分散在broker集群，leader partition也是均匀分散在broker集群

（8）延时处理机制：时间轮实现

生产端的架构原理

（1）元数据拉取机制：Topic -> Partition -> Leader & Follower -> Broker

（2）分区机制，默认的分区策略，hash分区策略

（3）缓冲区：数据结构，每个分区有多个batch

（4）Sender线程 + batch批量发送，按照broker来聚合多个batch作为request

（5）同步和异步

（6）核心参数：batch.size、缓冲区大小、acks、超时时间

消费端的架构原理

（1）消费模型：consumer group，每个分区就给一个consumer，queue & PUB/SUB

（2）分区分配机制：coordinator -> \_\_consumer\_offsets，JoinGroup -> leader consumer -> 下发分区方案 -> 状态机，三种分区策略

（2）消费方式：poll()调用，单线程干所有的事情

（3）offset内存数据结构更新，定时提交offset，内部topic（\_\_consumer\_offsets）

（4）故障感知机制：核心参数，心跳参数、会话超时、消费过慢

（5）老版本api的high-level和low-level，简单的认知就可以了，新版本

需求场景 -> 请求量 -> 数据量 -> 高峰并发量 -> 10亿消息

kafka集群部署：多少台物理机、磁盘规划（容量和类型）、内存规划（os cache & jvm）、CPU、网卡，10亿请求的场景，需要一个多大规模的kafka集群

kafka运维管理：支撑业务（topic创建和维护、可视化平台）、集群监控（可视化平台、开源工具）、资源使用率 -> 集群扩容 / topic扩容、机器负载率 -> 负载均衡、异常监控（报错 / jvm fullgc）、运维管理（多集群同步、安全认证、版本升级）

生产端机器部署：可能是你的java系统在生产消息，也可能是flume日志采集，canal数据库binlog采集，爬虫，如果你的机器就是每天1000万~2000万，数据往kafka里写入，没其他的数据库访问或者是别的线程和操作，4核8G的虚拟机要2台正好，高峰期的时候，CPU和内存的负载都到50%了，10亿 -> 40台4核8G虚拟机 -> 10台~20台左右16核32G的高配置虚拟机，也可以抗住每天生产10亿条数据的负载

消费端机器部署：flink、spark streaming、自己写的java系统，4核8G虚拟机，单线程消费出来 -> 线程池来处理（60），每秒钟消费吞吐量大概可以到个几百，一两千~两三千，对应的是说，他需要对数据库进行很多增删改查的操作，16核32G，每秒消费和处理5000条数据，10多台机器，spark streaming他每秒要消费和处理几万条数据，那么需要多少的资源，cpu core、内存，集群是运行在多少台机器上的集群，spark streaming，集群规模有30，480核 + 640G，160核 + 320G

kafka集群部署好，生产者代码写好 + 部署好，消费者代码写好 + 部署好，基本上已经可以run起来了，可以抗住每秒几万的并发请求，一天10亿的数据量流过kafka在处理，对其中的一些细节进行参数设置

数据保留几天，默认是7天，min.insync.replicas配合ISR + 数据不丢失，topic默认的分区数量，分区默认的副本数量，\_\_consumer\_offsets的分区数量和副本数量，processor线程数量，Handler线程池的大小，jvm参数的设置（堆大小 & G1垃圾回收器）

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/128\_目前为止对Kafka架构原理学习到的知识体系总结（三）：消费端/笔记.doc**

broker端的架构原理

（1）集群架构：Controller -> zk -> broker

（2）元数据管理：Controller，Topic -> Partition -> Leader & Follower -> Broker

（3）分布式架构：Topic逻辑数据集合 -> 物理上的多个Partition -> 分布在多台Broker

（4）broker端的多路复用模型的请求处理架构，自定义二进制协议

（5）磁盘存储，os cache，零拷贝，索引文件，日志文件，定期清理文件

（6）高可用架构：多副本冗余机制，Leader & Follower角色分配，leader -> follower的副本同步机制，LEO & HW的更新，ISR机制，Broker宕机后如何由Controller感知以及触发Leader重新选举

（7）负载均衡架构：所有partition物理的均匀分散在broker集群，leader partition也是均匀分散在broker集群

（8）延时处理机制：时间轮实现

生产端的架构原理

（1）元数据拉取机制：Topic -> Partition -> Leader & Follower -> Broker

（2）分区机制，默认的分区策略，hash分区策略

（3）缓冲区：数据结构，每个分区有多个batch

（4）Sender线程 + batch批量发送，按照broker来聚合多个batch作为request

（5）同步和异步

（6）核心参数：batch.size、缓冲区大小、acks、超时时间

消费端的架构原理

（1）消费模型：consumer group，每个分区就给一个consumer，queue & PUB/SUB

（2）分区分配机制：coordinator -> \_\_consumer\_offsets，JoinGroup -> leader consumer -> 下发分区方案 -> 状态机，三种分区策略

（2）消费方式：poll()调用，单线程干所有的事情

（3）offset内存数据结构更新，定时提交offset，内部topic（\_\_consumer\_offsets）

（4）故障感知机制：核心参数，心跳参数、会话超时、消费过慢

（5）老版本api的high-level和low-level，简单的认知就可以了，新版本

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/140\_目前架构下可能存在的问题（06）：消息的过期时间（TTL）如何实现/笔记.doc**

有的时候其实是希望对消息设置一个过期时间，TTL，就是如果在kafka里积压了一段时间，TTL过期时间，还没被消费和处理，那么就直接过期掉他，就不要继续去处理了，可以跟百万消息积压的方案结合起来来使用

比如说现在kafka里积压了100万消息，TTL（30秒），此时如果下游的消费者恢复了，可以继续消费了，此时他发现这些消息都到TTL了，就可以直接丢弃掉这些消息，不要慢吞吞的来处理

你可以把TTL + 死信队列来实现一套生产方案，过期的消息直接放到另外一个独立的特殊的队列里去，那个队列专门是用来处理一些长时间过期的消息的，有专门的后台的一套系统和机制来处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/143\_目前架构下可能存在的问题（09）：针对链路故障的死信队列应该如何实现/笔记.doc**

如果你的某个消息此时发送到kafka之后，比如说他在正常的topic里无法被处理，不管怎么说，下游的消费者整个就挂掉了，此时这些消息可以进入一个队列，特殊的队列，死信队列，处理失败的消息，放入特殊的队列中，后续可以用特殊的机制来进行处理

假设我们给消息设置了TTL，然后消息进入了kafka topic，结果下游消费者故障，导致百万消息积压了，消费者一旦恢复，对这些过期的消息，直接就不处理，放入到一个死信队列里去

后面可以用一些特殊的机制对死信队列中的消息进行处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/148\_目前架构下可能存在的问题（14）：对消息质量是否正常进行监控/笔记.doc**

你需要对所有的消息去进行一个监控汇总，每天一共流转过去多少条消息，10亿条，其中链路完整流转完整的消息占比多少，链路不完整处理错误的占比多少，诸如此类，你需要对你每天综合性的统计一下你的消息处理的情况

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/137\_目前架构下可能存在的问题（03）：MySQL binlog数据同步顺序为何不能出错/笔记.doc**

消息可能会乱序的一个问题

导致乱序的原因还是重试，<NodeId, Deque<Request>>，最多允许同时发送出去5个请求，忍受5个请求都发出去，但是没有收到响应，如果1和2两个请求都成功了，3这个请求失败了，4和5也成功了

对请求3重试，此时就算成功，他的数据也会排在4和5请求的后面了

消费者，一个consumer对应一个或者多个分区，分区内的消息一定是有顺序的，单线程消费各个分区的数据，所以单线程消费到的消息一定都是在分区内有顺序的，消费到的时候还是有顺序的

如果你把消息，每个消息提交一个任务给线程池异步处理，消息1~10分别分配给10个线程来并发的处理，但是此时可能先处理完消息10，然后处理完消息6，最后是处理完消息2，你的消息被处理的顺序完全不是按照顺序来的

比如你要从业务数据库，监听他的mysql binlog的变化，增删改，INSERT、UPDATE、DELETE相关的语句，对同一条数据的三个mysql bilog是：insert、update、delete。结果这个消息写入kafka是有顺序的

消费到了3条消息，提交给3个线程并发处理，他需要把这些操作同步到es里去，先在es里插入一条数据，然后更新，接着是删除，先执行delete，然后是update，接着是insert，最后反而es里有一条数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/131\_每天10亿消息的生产项目如何基于Kafka实现（三）：消费服务部署/笔记.doc**

broker端的架构原理

（1）集群架构：Controller -> zk -> broker

（2）元数据管理：Controller，Topic -> Partition -> Leader & Follower -> Broker

（3）分布式架构：Topic逻辑数据集合 -> 物理上的多个Partition -> 分布在多台Broker

（4）broker端的多路复用模型的请求处理架构，自定义二进制协议

（5）磁盘存储，os cache，零拷贝，索引文件，日志文件，定期清理文件

（6）高可用架构：多副本冗余机制，Leader & Follower角色分配，leader -> follower的副本同步机制，LEO & HW的更新，ISR机制，Broker宕机后如何由Controller感知以及触发Leader重新选举

（7）负载均衡架构：所有partition物理的均匀分散在broker集群，leader partition也是均匀分散在broker集群

（8）延时处理机制：时间轮实现

生产端的架构原理

（1）元数据拉取机制：Topic -> Partition -> Leader & Follower -> Broker

（2）分区机制，默认的分区策略，hash分区策略

（3）缓冲区：数据结构，每个分区有多个batch

（4）Sender线程 + batch批量发送，按照broker来聚合多个batch作为request

（5）同步和异步

（6）核心参数：batch.size、缓冲区大小、acks、超时时间

消费端的架构原理

（1）消费模型：consumer group，每个分区就给一个consumer，queue & PUB/SUB

（2）分区分配机制：coordinator -> \_\_consumer\_offsets，JoinGroup -> leader consumer -> 下发分区方案 -> 状态机，三种分区策略

（2）消费方式：poll()调用，单线程干所有的事情

（3）offset内存数据结构更新，定时提交offset，内部topic（\_\_consumer\_offsets）

（4）故障感知机制：核心参数，心跳参数、会话超时、消费过慢

（5）老版本api的high-level和low-level，简单的认知就可以了，新版本

需求场景 -> 请求量 -> 数据量 -> 高峰并发量 -> 10亿消息

kafka集群部署：多少台物理机、磁盘规划（容量和类型）、内存规划（os cache & jvm）、CPU、网卡，10亿请求的场景，需要一个多大规模的kafka集群

kafka运维管理：支撑业务（topic创建和维护、可视化平台）、集群监控（可视化平台、开源工具）、资源使用率 -> 集群扩容 / topic扩容、机器负载率 -> 负载均衡、异常监控（报错 / jvm fullgc）、运维管理（多集群同步、安全认证、版本升级）

生产端机器部署：可能是你的java系统在生产消息，也可能是flume日志采集，canal数据库binlog采集，爬虫，如果你的机器就是每天1000万~2000万，数据往kafka里写入，没其他的数据库访问或者是别的线程和操作，4核8G的虚拟机要2台正好，高峰期的时候，CPU和内存的负载都到50%了，10亿 -> 40台4核8G虚拟机 -> 10台~20台左右16核32G的高配置虚拟机，也可以抗住每天生产10亿条数据的负载

消费端机器部署：flink、spark streaming、自己写的java系统，4核8G虚拟机，单线程消费出来 -> 线程池来处理（60），每秒钟消费吞吐量大概可以到个几百，一两千~两三千，对应的是说，他需要对数据库进行很多增删改查的操作，16核32G，每秒消费和处理5000条数据，10多台机器，spark streaming他每秒要消费和处理几万条数据，那么需要多少的资源，cpu core、内存，集群是运行在多少台机器上的集群，spark streaming，集群规模有30，480核 + 640G，160核 + 320G

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/125~151资料/133\_基于对Kafka架构原理的理解合理设置各种参数（二）：生产服务参数/笔记.doc**

broker端的架构原理

（1）集群架构：Controller -> zk -> broker

（2）元数据管理：Controller，Topic -> Partition -> Leader & Follower -> Broker

（3）分布式架构：Topic逻辑数据集合 -> 物理上的多个Partition -> 分布在多台Broker

（4）broker端的多路复用模型的请求处理架构，自定义二进制协议

（5）磁盘存储，os cache，零拷贝，索引文件，日志文件，定期清理文件

（6）高可用架构：多副本冗余机制，Leader & Follower角色分配，leader -> follower的副本同步机制，LEO & HW的更新，ISR机制，Broker宕机后如何由Controller感知以及触发Leader重新选举

（7）负载均衡架构：所有partition物理的均匀分散在broker集群，leader partition也是均匀分散在broker集群

（8）延时处理机制：时间轮实现

生产端的架构原理

（1）元数据拉取机制：Topic -> Partition -> Leader & Follower -> Broker

（2）分区机制，默认的分区策略，hash分区策略

（3）缓冲区：数据结构，每个分区有多个batch

（4）Sender线程 + batch批量发送，按照broker来聚合多个batch作为request

（5）同步和异步

（6）核心参数：batch.size、缓冲区大小、acks、超时时间

消费端的架构原理

（1）消费模型：consumer group，每个分区就给一个consumer，queue & PUB/SUB

（2）分区分配机制：coordinator -> \_\_consumer\_offsets，JoinGroup -> leader consumer -> 下发分区方案 -> 状态机，三种分区策略

（2）消费方式：poll()调用，单线程干所有的事情

（3）offset内存数据结构更新，定时提交offset，内部topic（\_\_consumer\_offsets）

（4）故障感知机制：核心参数，心跳参数、会话超时、消费过慢

（5）老版本api的high-level和low-level，简单的认知就可以了，新版本

需求场景 -> 请求量 -> 数据量 -> 高峰并发量 -> 10亿消息

kafka集群部署：多少台物理机、磁盘规划（容量和类型）、内存规划（os cache & jvm）、CPU、网卡，10亿请求的场景，需要一个多大规模的kafka集群

kafka运维管理：支撑业务（topic创建和维护、可视化平台）、集群监控（可视化平台、开源工具）、资源使用率 -> 集群扩容 / topic扩容、机器负载率 -> 负载均衡、异常监控（报错 / jvm fullgc）、运维管理（多集群同步、安全认证、版本升级）

生产端机器部署：可能是你的java系统在生产消息，也可能是flume日志采集，canal数据库binlog采集，爬虫，如果你的机器就是每天1000万~2000万，数据往kafka里写入，没其他的数据库访问或者是别的线程和操作，4核8G的虚拟机要2台正好，高峰期的时候，CPU和内存的负载都到50%了，10亿 -> 40台4核8G虚拟机 -> 10台~20台左右16核32G的高配置虚拟机，也可以抗住每天生产10亿条数据的负载

消费端机器部署：flink、spark streaming、自己写的java系统，4核8G虚拟机，单线程消费出来 -> 线程池来处理（60），每秒钟消费吞吐量大概可以到个几百，一两千~两三千，对应的是说，他需要对数据库进行很多增删改查的操作，16核32G，每秒消费和处理5000条数据，10多台机器，spark streaming他每秒要消费和处理几万条数据，那么需要多少的资源，cpu core、内存，集群是运行在多少台机器上的集群，spark streaming，集群规模有30，480核 + 640G，160核 + 320G

kafka集群部署好，生产者代码写好 + 部署好，消费者代码写好 + 部署好，基本上已经可以run起来了，可以抗住每秒几万的并发请求，一天10亿的数据量流过kafka在处理，对其中的一些细节进行参数设置

数据保留几天，默认是7天，min.insync.replicas配合ISR + 数据不丢失，topic默认的分区数量，分区默认的副本数量，\_\_consumer\_offsets的分区数量和副本数量，processor线程数量，Handler线程池的大小，jvm参数的设置（堆大小 & G1垃圾回收器）

生产端参数：缓冲区、batch、超时时间、acks

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/057\_测试kafka集群的topic创建、消息的生成以及消费/笔记.doc**

bin/kafka-topics.sh --zookeeper hadoop03:2181,hadoop04:2181,hadoop05:2181 --create --topic test-topic --partitions 3 --replication-factor 2

bin/kafka-topics.sh --zookeeper hadoop03:2181,hadoop04:2181,hadoop05:2181 -list

0,1,2，说的就是broker.id

bin/kafka-topics.sh --zookeeper hadoop03:2181,hadoop04:2181,hadoop05:2181 --describe --topic test-topic

bin/kafka-console-producer.sh --broker-list hadoop03:9092,hadoop04:9092,hadoop05:9092 --topic test-topic

bin/kafka-console-consumer.sh --bootstrap-server hadoop03:9092,hadoop04:9092,hadoop05:9092 --topic test-topic

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/056\_基于生产参数部署3个节点组成的Kafka集群/笔记.doc**

tar -zxvf kafka\_2.11-1.0.0.tgz

mv kafka\_2.11-1.0.0 kafka

mkdir -p /usr/kafka/data

cd kafka

vi server.properties

broker.id=0

listeners=PLAINTEXT://hadoop01:9092

log.dirs=/usr/kafka/data

zookeeper.connect=hadoop01:2181,hadoop02:2181,hadoop03:2181

zookeeper.connection.timeout.ms=6000

bin/kafka-server-start.sh.sh -daemon config/server.properties

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/049\_Kafka集群生产规划：千兆网卡还是万兆网卡？/笔记.doc**

10亿数据，6w QPS，12TB，6台物理机，2块1TB的机械硬盘，64GB内存（6GB给JVM，剩余给os cache），16核CPU，千兆网卡

生产环境的机器规划，无论是MySQL、Redis、RocketMQ、Elasticsearch、Kafka、Hadoop、Flink，Yarn，规划的思路都是类似的，从技术本质和底层的原理出发来考虑，请求量有多大、数据量有多大、内存应该分配多大（底层工作机制）、线程数量有多少、网络数据传输量有多大

现在一般就是千兆网卡（1GB / s），还有万兆网卡（10GB / s）

你要计算一下，kafka集群之间，broker和broker之间是会做数据同步的，因为leader要同步数据到follower上去，他们是在不同的broker机器上的，broker机器之间会进行频繁的数据同步，传输大量的数据

每秒两台broker机器之间大概会传输多大的数据量？高峰期每秒大概会涌入6万条数据，每秒就是60mb/s的数据量，他会在broker之间来传输，你就算多估算一些，每秒大概也就传输个几百mb/s的数据，就足够了

针对千兆网卡来算一下，假如每台物理机的网卡带宽都是1GB / s

实际上每台机器能用的网卡的带宽还达不到极限，因为kafka只能用其中一部分的带宽资源，比如700mb / s，但是一般不能允许kafka占用这么多带宽，因为避免说占用这么多带宽，万一再多一点，就容易把网卡打满

所以说，一般限制kafka每秒带宽资源就是300mb / s，如果你给物理机使用的千兆网卡，那么其实他每秒最多传输数据是几百mb，是够了，几十mb，或者一两百mb，基本上都够了，可以足够传输

假如说现在每秒要传输7万条数据，每条数据是平均1kb，那么就是70mb / s，这样的话，是可以轻松抗住的

如果是万兆网卡，那么就更加轻松了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/043\_Kafka集群生产规划：需要部署几台物理机？/笔记.doc**

10亿数据，高峰期6万QPS，集群容纳的数据量是13TB

部署Kafka，Hadoop，MySQL，大数据核心分布式系统，一般建议大家直接采用物理机，不建议用一些低配置的虚拟机自己瞎搞

QPS这个东西，不可能是说，你只要支撑6万QPS，你的集群就刚好支撑6万QPS就可以了。假如说你只要支撑6w QPS，2台物理机绝对绝对够了，单台物理机部署kafka支撑个几万QPS是没问题的

但是这里有一个问题，我们通常是建议，公司预算充足，尽量是让高峰QPS控制在集群能承载的总QPS的30%左右

你的kafka集群能承载的总QPS给他搞成20万~30万，是非常安全的

大体上来说，需要5~7台物理机来部署，基本上就很安全了，每台物理机要求吞吐量在每秒几万条数据就可以了，物理机的配置和性能也不需要特别高

硬盘容量这块，每台物理机硬盘空间起码有个几个TB，所以如果是6台物理机的话，起码可以支撑个十几TB的数据，你到底需要不需要保留最近7天的数据，也可以根据你的需求场景减小这个时间间隔

保留最近3天的数据，6TB左右

如果你kafka集群的硬盘容量总共加起来有个十几TB，或者是20TB，都可以来支撑

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/053\_生产环境中的Kafka集群参数设置：操作系统参数/笔记.doc**

文件描述符限制：kafka会频繁的创建和修改文件，大约是分区数量 \* （分区总大小 / 日志段大小） \* 3，比如一个broker上大概有100个分区，每个分区大概是10G的数据，日志段大小是默认的1G，那么就是100 \* 10（分区里的日志段文件数量） \* 3 = 3000个文件

ulimit -n 100000，这个可以设置很大的描述符限制，允许创建大量的文件

磁盘flush时间：默认是5秒，os cache刷入磁盘，可以设置为几分钟，比如1分钟才刷入磁盘，这样可以大幅度提升单机的吞吐量

sysctl -a | grep dirty

vm.dirty\_background\_bytes = 0

vm.dirty\_background\_ratio = 5

vm.dirty\_bytes = 0

vm.dirty\_expire\_centisecs = 3000

vm.dirty\_ratio = 10

vm.dirty\_writeback\_centisecs = 500

vm.dirty\_writeback\_centisecs：每隔5秒唤醒一次刷磁盘的线程

vm.dirty\_expire\_centisecs：os cache里的数据在3秒后会被刷入磁盘

可以设置大一些，1分钟~2分钟都可以，特别大规模的企业和公司，如果你可以接收机器宕机的时候，数据适当可以丢失一些，kafka里的数据可以适当丢失一些，但是为了提升集群的吞吐量的话

大数据分析类的应用的话，主要是出一些数据报表，不要求数据100%精准的话，允许适当丢一些数据，此时的话，这里给他调大一些是没问题的

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/044\_Kafka集群生产规划：到底该不该用SSD？/笔记.doc**

10亿，6w/s的吞吐量，12TB的数据量，6台物理机，2块1TB的SAS盘

CPU、内存、硬盘、网路、参数，各种环节应该如何来规划

SSD固态硬盘，还是普通机械硬盘

比如说我们在规划和部署线上系统的MySQL集群的时候，一般来说必须用SSD性能可以提高很多，MySQL可以承载的并发请求量也会高很多，而且SQL语句执行的性能也会提高很多

SSD，固态硬盘，快，比机械硬盘要快，到底是快在哪里呢？快主要是快在磁盘随机读写，就要对磁盘上的随机位置来读写的时候，SSD比机械硬盘要快。像比如说是MySQL这种系统，就应该使用SSD了

Kafka集群，物理机是用昂贵的SSD呢？还是用普通的机械硬盘呢？写磁盘的时候，他是怎么来写的？磁盘顺序写的，压测，机械硬盘顺序写的性能机会跟内存读写的性能是差不多的，SSD做顺序写的性能

SSD主要是随机读写的性能要比机械硬盘随机读写的性能高很多倍

他在使用硬盘的时候，最核心的一个思想就是仅仅允许追加数据在每个日志段文件的末尾，不支持在磁盘文件里随机写，磁盘随机写

其实就没必要使用SSD了，会贵很多，增加很大的机器成本；顺序写，每台物理机给配置个常规的2块机械硬盘就可以，就足够了

每天多少条消息，每条消息几个备份，保留几天的数据，总共需要多大容量，对应到每台机器需要多大容量

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/041\_每日10亿数据会对Kafka集群造成多大压力/笔记.doc**

电商平台假设有每日10亿数据，此时我们的生产环境的kafka集群应该如何来规划和部署

10亿数据，他对kafka集群造成的并发请求会有多少，24小时，电商平台，其中有12个小时，是承担了大部分的流量，比如说晚上12点过后，到第二天的早上8点，这之间可能没什么流量过来

16小时 -> 有数据的，但是其中高峰期，可能就20%的时间，`16 \* 0.2 = 3小时

8亿，80%的数据会集中在16小时内涌入，8亿时间里的80%的请求会集中在3小时的高峰时间段，8 \* 0.8 = 6.4亿数据

大概是每秒钟有6万请求，会涌入你的kafka集群，整体kafka的集群需要抗住高峰期每秒6万的QPS

kafka集群默认是保留最近7天的数据，假设我们就用默认的配置

每天是进来10亿条数据，但是每条数据做几个副本？每个topic都可以设置副本因子，平均都是2个副本，每天在kafka磁盘上需要保留10亿\* 2 = 20亿条数据，集群磁盘空间需要保留最近7天的20亿 \* 7 = 140亿条数据

每条数据大概有多大？算大点，平均每条数据是1kb，140亿kb -> 整个kafka集群的磁盘空间至少需要容纳13TB的数据，就是最近7天的数据量

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/050\_Kafka集群生产规划：线上机器典型配置/笔记.doc**

10亿数据，6w/s的吞吐量，12TB的数据量

6台物理机

硬盘：2块机械硬盘，每块1TB，7200转

内存：64GB，JVM分配6G，剩余的给os cache

CPU：16核

网络：千兆网卡

下周的课，就可以来玩儿一下集群的管理和监控，加broker机器进去，给topic增加partition，给partition增加副本，对kafka集群进行升级，诸如此类的一些日常的运维管理的操作，对整个集群从各个角度如何来进行监控

一周的课，把生产和消费的客户端的机制、原理以及一般如何使用可以说一下

架构都跑通了以后，咱们就可以来玩儿一下kafka生产调优的实践，生产环境常见的一些问题，kafka的数据保证顺序，如何保证100%数据是零丢失的，如何解决线上百万数据积压的一个问题，还有就是如何对生产端、消费端、broker端进行吞吐量、性能、延迟的调优，提升整体kafka集群的读和写的性能

用几周的课，来学一下kafka的源码设计，把源码级别的东西给扣一下，可以让大家以后出去在面试，或者工作里用到kafka的时候，就可以很好的从底层的角度来面试，或者是解决自己日常的一些问题

三个端的数据采集，用户行为日志、数据库binlog、竞对数据爬取，写入到flume和kafka里面去

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/052\_生产环境中的Kafka集群参数设置：JVM以及GC参数/笔记.doc**

修改bin/kafka-start-server.sh中的jvm设置

export KAFKA\_HEAP\_OPTS=”-Xmx6g -Xms6g -XX:MetaspaceSize=96m -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:G1HeapRegionSize=16M -XX:MinMetaspaceFreeRatio=50 -XX:MaxMetaspaceFreeRatio=80”

大数据这块，基本上全部都是JVM体系，java、scala，全部都是java虚拟机，一般来说你无论是玩儿什么技术，都涉及到JVM这块的知识，需要进行设置，调优，垃圾回收，fullgc卡顿，jvm监控

包括他里面的jvm原理，他的线上问题处理，大量的案例和case组成的

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/054\_基于VirtualBos准备3台linux虚拟机模拟线上生产环境/笔记.doc**

hadoop环境，自己来看看环境是否还ok

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/046\_Kafka集群生产规划：充分利用os cache提高性能/笔记.doc**

10亿数据，6w QPS，12TB，6台物理机，2块1TB的机械硬盘

接下来看看，内存这块到底应该怎么来弄，kafka部署的时候内存的规划是很有讲究的，你一定要明白他底层的原理，才能知道应该如何来规划kafka的内存空间

kafka写数据基本上优先让数据写入os cache，后面来消费数据的时候，就可以从os cache里读取数据出来，写和读基本上都是依托于操作系统的缓存来执行的，我们在规划这个内存空间的时候

kafka本身是scala语言开发的，是基于jvm的，jvm代表了进程使用的内存空间，是多少，jvm堆内存，机器上留给os cache的内存应该有多大。kafka jvm堆内存，你觉得需要很大吗？你其实听懂了kafka原理之后

kafka并没有在自己的jvm堆内存里放入过多的数据，rabbitmq是不一样的，数据过来优先写入jvm堆内存里去缓冲一下，一定时间之后，rabbitmq再一次性把jvm堆内存里的缓冲一批数据给刷入磁盘中

导致jvm堆内存中存放大量的数据，需要给jvm堆内存开辟比较大的空间了

但是kafka正好相反的，他接收到了这个数据之后不是直接写jvm堆内存的，而是采用自己的二进制紧凑的数据格式，给写入到磁盘文件里去，是先写入os cache（操作系统管理的一块内存缓冲空间）

kafka并没有使用过多的jvm堆内存空间，不需要给kafka jvm堆内存分配过大的空间，基本上来说几个G就够了

在kafka部署的机器上，你需要比较大的空间应该是os cache这块，如果这块空间越大，那么你其实就可以在os cache内存里存放越多的数据，就可以保证说，人家在消费的时候，就可以从os cache里去读取数据了

os cache这块空间起码要有多大，在底层磁盘文件这块，其实是每个分区的数据是分段存的，每个分区的数据有多个分段日志文件，人家消费的时候比较频繁的需要读取的，一般是最新的这个正在写入的分段日志文件

起码你应该让这台机器上的broker管理的每个parition的最新正在写的日志段文件的数据都可以驻留在os cache中，这样可以保证每个parition正在写的数据，最有可能被消费的数据，就可以直接从os cache里来读了

假设100个topic，每个topic是6个partition，每个partiton是2个副本，一共100 \* 6 =600个leader partition，平均到6台机器上去，假设每台机器是放100个partition，每个partition的最新正在写的日志段文件的大小是默认的1GB

所以说单台机器上，最新的正在写的日志段文件的大小100个partition \* 1GB的日志段文件 = 600GB。如果最佳的情况，单台机器可以有600GB的内存的给os cache的话，就可以每个partiton最新的数据都在os cache里，比如说一个日志段文件有1GB，可能都对应几千万条数据了

没必要说一定要几千万的数据都在os cache里才可以，日志段文件3000万条数据，其实人家消费比较频繁的，就是最近的10万条数据，2GB的数据可以保留在os cache中，最新写入的2GB的数据驻留在os cache中，就可以让人家从内存里消费到最新的数据了

给os cache可以分配的内存空间绝对是不止2GB，几十个GB，每个partition的最新日志段文件里的最新写入的10%的数据，都可以驻留在os cache中，让人家对最新的数据都可以从os cache里读取到

你必须保证最新的10%的数据都在os cache里，人家消费大部分的消费请求可以走os cache来读取就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/048\_Kafka集群生产规划：为什么需要16核CPU？/笔记.doc**

10亿数据，6w QPS，12TB，6台物理机，2块1TB的机械硬盘，64GB内存（6GB给JVM，剩余给os cache），16核CPU

CPU规划，主要是看你的这个进程里会有多少个线程，线程主要是依托多核CPU来执行的，如果你的线程特别多，但是CPU核很少，就会导致你的CPU负载很高，会导致你的整体工作线程执行的效率不太高

acceptor线程负责去接入客户端的连接请求，但是他接入了之后其实就会把连接分配给多个processor，默认是3个，但是说实话一般生产环境的话呢 ，建议大家还是多加几个，整体可以提升kafka的吞吐量

比如说你可以增加到6个，或者是9个

另外就是负责处理请求的线程，是一个线程池，默认是8个线程，在生产集群里，建议大家可以把这块的线程数量稍微多加个2倍~3倍，其实都正常，比如说搞个16个工作线程，24个工作线程

他后台会有很多的其他的一些线程，比如说定期清理7天前数据的线程，Controller负责感知和管控整个集群的线程，后台线程在工作

每个broker可能都会有上百个起码，一两百个线程繁忙的在工作

CPU给到4核，一般来说几十个线程，在高峰期CPU几乎都快打满了，8核，也就能够比较宽裕的支撑几十个线程繁忙的工作，一般是建议16核，靠谱，基本上可以hold住一两百线程的工作

broker一般都会启动几十个甚至上百个线程，大家看过broker端的原理了，各种处理请求的线程，后台线程，几十个线程频繁工作，一般都建议是16核CPU，甚至32核CPU

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/045\_Kafka集群生产规划：硬盘空间应该给多大？/笔记.doc**

10亿数据，6w QPS，12TB，6台物理机，2块1TB的机械硬盘

最近7天一共是12TB的数据量的角度而言，大概就是每台物理机搞2块机械硬盘，每块盘就是1TB好像就够了

很多是从mysql同步的binlog数据过来的，那种数据很多是到不了1kb的，几十个字节，几百个字节，也算是比较大了，用户行为日志，也不一定就可以到达1kb一条，一两百个字节，具体看你们公司采集的数据，看一看，估算一下平均每条数据有多大

有一个经验值，1亿条用户行为日志，也就几个GB，几个TB，也就差不多了

未来给机器扩容硬盘也是可以的，继续加硬盘，或者把小容量的硬盘换成大容量的盘，运维工程师会协助你来做的，偏硬件一些的，给服务器加硬盘，挂载，等等，诸如此类的一些事情

很多公司，每天其实就几千万条数据采集到大数据平台里来的话，根本不算是什么大数据，几百MB，1GB以内，没多少数据量

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/051\_生产环境中的Kafka集群参数设置：内核参数/笔记.doc**

自己可能都或多或少哪怕是虚拟机环境的kafka都搭建过

broker.id=0，这个是每个broker都必须自己设置的一个唯一id

log.dirs，这个极为重要，kafka的所有数据就是写入这个目录下的磁盘文件中的，如果说机器上有多块物理硬盘，那么可以把多个目录挂载到不同的物理硬盘上，然后这里可以设置多个目录，这样kafka可以数据分散到多块物理硬盘，多个硬盘的磁头可以并行写，这样可以提升吞吐量

zookeeper.connect，同样很重要，这个是连接kafka底层的zookeeper集群的

listeners，这是broker监听客户端发起请求的端口号

unclean.leader.election.enable，默认是false，意思就是只能选举ISR列表里的follower成为新的leader，1.0版本后才设为false，之前都是true，允许非ISR列表的follower选举为新的leader

delete.topic.enable，默认true，允许删除topic

log.retention.hours，可以设置一下，要保留数据多少个小时，这个就是底层的磁盘文件，默认保留7天的数据，根据自己的需求来就行了

log.retention.bytes，这个设置如果分区的数据量超过了这个限制，就会自动清理数据，默认-1不按照这个策略来清理，这个一般不常用

min.insync.replicas，这个跟acks=-1配合起来使用，意思就是说必须要求ISR列表里有几个follower，然后acks=-1就是写入数据的时候，必须写入这个数量指定的follower才可以，一般是可以考虑如果一个leader两个follower，三个副本，那么这个设置为2，可以容忍一台机器宕机，保证高可用，但是写入数据的时候必须ISR里有2个副本，而且必须副本都写入才算成功

如果你就一个leader和follower，双副本，min.insync.replicas = 2。如果有一台机器宕机，导致follower没了，此时ISR列表里就一个leader的话，你就不能写入了，如果你只有一个副本了，此时还可以写入数据的话，就会导致写入到leader之后，万一leader也宕机了，此时数据必然丢失

一般来说为了避免数据占用磁盘空间过大，一般都是kafka设置双副本，一个leader和一个follower就够了，设置的三副本的话，数据在集群里乘以3倍的空间来存放，非常耗费磁盘空间，对你的整体集群的性能消耗也会更大

min.insync.replicas=1，acks=-1（一条数据必须写入ISR里所有副本才算成功），你写一条数据只要写入leader就算成功了，不需要等待同步到follower才算写成功。但是此时如果一个follower宕机了，你写一条数据到leader之后，leader也宕机，会导致数据的丢失。

一般来说双副本的场景下，我们为了避免数据丢失，min.insync.replicas=2，acks=-1，每次写入必须保证ISR里2个副本都写成功才可以，如果其中一个副本没了，会导致你就没法写入，必须阻塞等待kafka恢复

这样就可以保证数据的不丢失

你是要计算为客户准备的财务数据报表，非常严格的数据必须精准的话，精准到每一分钱，你还是得设置成2

num.network.threads，这个是负责转发请求给实际工作线程的网络请求处理线程的数量，默认是3，高负载场景下可以设置大一些

num.io.threads，这个是控制实际处理请求的线程数量，默认是8，高负载场景下可以设置大一些

要自己做一些压测，在不同的线程数量的条件下，对整体的生产和消费的吞吐量分别是多少，还得看一下线程越多的时候，对CPU的复杂有多大，整体看一下，如果增加一些线程，吞吐量提升很多，而且CPU负载还能接受

message.max.bytes，这个是broker默认能接受的消息的最大大小，默认是977kb，太小了，可以设置的大一些，一般建议设置大一些，很多大消息可能大到几mb，这个设置个10mb，可以考虑

log.flush.interval.messages

log.flush.interval.ms

线上的常用的一个配合，高负载高吞吐的情况下，建议设置为1分钟

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/058\_对虚拟机组成的Kafka集群测试其生产和消费的吞吐量/笔记.doc**

bin/kafka-producer-perf-test.sh --topic test-topic --num-records 500000 --record-size 200 --throughput -1 --producer-props bootstrap.servers=hadoop03:9092,hadoop04:9092,hadoop05:9092 acks=-1

bin/kafka-consumer-perf-test.sh --broker-list hadoop03:9092,hadoop04:9092,hadoop53:9092 --fetch-size 2000 --messages 500000 --topic test-topic

哪怕是非常挫的虚拟机部署的kafka三个节点的集群，居然都可以做到每秒发送1万条数据，每秒消费接近2万条数据，kafka架构 设计优秀，性能压榨到了极致，可以做到，非常高的吞吐来使用kafka的集群

在我们说的那种物理机的配置之下，单机每秒10万+的数据量是没问题的，6台物理机，整个集群抗住每秒几十万的请求，是ok的，是没问题的，每天10亿条数据，是绰绰有余的，就会导致集群不够用

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/041~058资料/047\_Kafka集群生产规划：内存空间如何规划？/笔记.doc**

kafka自身的jvm是用不了过多堆内存的，因为kafka设计就是规避掉用jvm对象来保存数据，避免频繁fullgc导致的问题，所以一般kafka自身的jvm堆内存，分配个6G左右就够了，剩下的内存全部留给os cache

32G以上的内存，尽量是64G~128G的内存空间

比如一台物理机，有64GB的内存空间，那么50多G都是给os cache的

然后要看看每个日志段的大小，每个分区都有一个日志段是当前正在写入的，这个大家都懂了，所以此时这个正在写入日志段的数据，就是消费者读取最频繁的数据，主要让这台机器上每个partition副本的正在写的日志段，都可以放在os cache里

那么基本上80%~90%的消费行为，都可以直接从os cache里读到数据了，如果你真的按照课程的思路和讲解，来推算和规划你的kafka生产集群，把的内存规划设置的很好的话，基本上可以做到

整个人家kafka的读写吞吐量都很大，性能很高，延迟很低

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/112\_自行指定每个Consumer要消费哪些分区有用吗？/笔记.doc**

List<PartitionInfo> partitions = consumer.partitionsFor(“order-topic”);

new TopicPartition(partitionInfo.topic(), partitionInfo.partition());

consumer.assign(partitions); //指定每个consumer要消费哪些分区，你就不是依靠consumer的自动的分区分配方案来做了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/120\_Kafka自定义的基于TCP的二进制协议深入探秘一番（二）/笔记.doc**

比如说发送消息，就是ProduceRequest和ProduceResponse，代表“PRODUCE”这个接口的请求和响应，api\_key=0，其实就是“PRODUCE”接口的代表

他的RequestBody，包含了：transactional\_id，acks，timeout，topic\_data（topic，data（partition，record\_set）），acks就是客户端自己指定的acks参数，这个会指示leader和follower副本的写入方式，timeout就是超时时间，默认就是30秒，request.timeout.ms

然后就是要写入哪个topic，哪个分区，以及对应数据集合，里面是多个batch

ProduceResponse，ResponseBody，包含了responses（topic，partition\_responses（partition，error\_code，base\_offset，log\_append\_time，log\_start\_offset）），throttle\_time\_ms，简单来说就是当前响应是对哪个topic写入的响应

包含了每个topic的各个分区的响应，每个partition的写入响应，包括error\_code错误码，base\_offset是消息集合的起始offset，log\_append\_time是写入broker端的时间，log\_start\_offset是分区的起始offset

其实各种接口大体上来说就是如此，所以现在大家就知道了，协议就是一种规定，你发送过来的请求是什么格式的，他可能有请求头还有请求体，分别包含哪些字段，按什么格式放数据，响应也是一样的

然后大家就可以按一样的协议来发送请求和接收响应

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/101\_Group Coordinator是什么以及主要负责什么？/笔记.doc**

生产者和消费者的基本原理，使用，核心参数，这一周的课，我们就是对kafka的生产者、消费者、协议，底层的原理做一些更加深入的理解

每个consumer group都会选择一个broker作为自己的coordinator，他是负责监控这个消费组里的各个消费者的心跳，以及判断是否宕机，然后开启rebalance的，那么这个如何选择呢？

就是根据group.id来进行选择，他有内部的一个选择机制，会给你挑选一个对应的Broker，总会把你的各个消费组均匀分配给各个Broker作为coordinator来进行管理的

他负责的事情只要就是rebalance，说白了你的consumer group中的每个consumer刚刚启动就会跟选举出来的这个consumer group对应的coordinator所在的broker进行通信，然后由coordinator分配分区给你的这个consumer来进行消费

coordinator会尽可能均匀的分配分区给各个consumer来消费

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/118\_同时可以接受有几个发送到Broker的请求没收到响应？/笔记.doc**

Map<NodeId, Deque<Request>> => 给这个broker发送了哪些请求过去了？

max.in.flight.requests.per.connection：5

这个参数默认值是5，默认情况下，每个Broker最多只能有5个请求是发送出去但是还没接收到响应的，所以这种情况下是有可能导致顺序错乱的，大家一定要搞清楚这一点，先发送的请求可能后续要重发

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/109\_如何实现Consumer Group的状态机流转机制？/笔记.doc**

刚开始Consumer Group状态是：Empty

接着如果部分consumer发送了JoinGroup请求，会进入：PreparingRebalance的状态，等待一段时间其他成员加入，这个时间现在默认就是max.poll.interval.ms来指定的，所以这个时间间隔一般可以稍微大一点

接着如果所有成员都加入组了，就会进入AwaitingSync状态，这个时候就不能允许任何一个consumer提交offset了，因为马上要rebalance了，进行重新分配了，这个时候就会选择一个leader consumer，由他来制定分区方案

然后leader consumer制定好了分区方案，SyncGroup请求发送给coordinator，他再下发方案给所有的consumer成员，此时进入stable状态，都可以正常基于poll来消费了

所以如果说在stable状态下，有consumer进入组或者离开崩溃了，那么都会重新进入PreparingRebalance状态，重新看看当前组里有谁，如果剩下的组员都在，那么就进入AwaitingSync状态

leader consumer重新制定方案，然后再下发

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/105\_Consumer内部单线程处理一切事务的核心设计思想/笔记.doc**

其实就是在一个while循环里不停的去调用poll()方法，其实是我们自己的一个线程，就是我们自己的这个线程就是唯一的KafkaConsumer的工作线程，新版本的kafka api，简化，减少了线程数量

Consumer自己内部就一个后台线程，定时发送心跳给broker；但是其实负责进行拉取消息、缓存消息、在内存里更新offset、每隔一段时间提交offset、执行rebalance这些任务的就一个线程，其实就是我们调用Consumer.poll()方法的那个线程

就一个线程调用进去，会负责把所有的事情都干了

为什么叫做poll呢？因为就是你可以监听N多个Topic的消息，此时会跟集群里很多Kafka Broker维护一个Socket连接，然后每一次线程调用poll()，就会监听多个socket是否有消息传递过来

可能一个consumer会消费很多个partition，每个partition其实都是leader可能在不同的broker上，那么如果consumer要拉取多个partition的数据，就需要跟多个broker进行通信，维护socket

每个socket就会跟一个broker进行通信

每个Consumer内部会维护多个Socket，负责跟多个Broker进行通信，我们就一个工作线程每次调用poll()的时候，他其实会监听多个socket跟broker的通信，是否有新的数据可以去拉取

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/124\_Kafka的时间轮延时调度机制（三）：时间轮层级的下滑/笔记.doc**

按照之前说的，假如说有一个定时450ms的任务，会被放到第三层时间轮的第一个时间格，他的时间是400ms，那么这个时间轮的指针转到这里，发现这个任务其实还有50ms才到期，此时就会把他重新放回到第二层时间轮

第一层时间轮：1ms \* 20

第二层时间轮：20ms \* 20

第三层时间轮：400ms \* 20

对于第三层时间轮而言，每一个格子都是400ms，对于450ms的任务，就放在第三层时间轮的第二个格子里就好了，假设说currentTime指向的是第一个格子，然后400ms以后就会走到第二个格子，此时400ms已经过去了

发现这个任务是450ms以后才执行的，此时还有50ms才可以执行这个任务

降层，第二层时间轮，20ms，第一个格子是0，第二个格子是20ms，第三个格子是40ms，所以此时还剩50ms的任务可以放在第二层时间轮的第三个格子里，第二层时间轮转到第三个格子时，40ms过去了，还剩10ms

降层，第一层时间轮，1ms，放在第10个格子里就可以了，10ms过后指针转过去就可以执行这个延时任务了

进入第二层时间轮的距离currentTime两个时间格的地方，第二层时间轮的currentTime转过两个时间格之后，也就是40ms后发现这个任务，其实剩余时间还有10ms，就会降低到第一层时间轮，距离currentTime有10个时间格的地方

第一层时间轮的currentTime转过10个时间格之后，也就是10ms之后，就会执行这个任务

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/125\_Kafka的时间轮延时调度机制（四）：基于DelayQueue推动/笔记.doc**

基于数组和双向链表来O(1)时间度可以插入任务

但是推进时间轮怎么做呢？搞一个线程不停的空循环判断是否进入下一个时间格吗？那样很浪费CPU资源，所以采取的是DelayQueue

每个时间轮里的TimerTaskList作为这个时间格的任务列表，都会插入DelayQueue中，设置一个延时出队时间，DelayQueue会自动把过期时间最短的排在队头，然后专门有一个线程来从DelayQueue里获取到期任务列表

某个时间格对应的TimerTaskList到期之后，就会被线程获取到，这种方式就可以实现时间轮推进的效果，推进时间轮基于DelayQueue，时间复杂度也是O(1)，因为只要从队头获取即可

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/119\_Kafka自定义的基于TCP的二进制协议深入探秘一番（一）/笔记.doc**

kafka自定义了一组二进制的协议，现在一共是包含了43种协议类型，每种协议都有对应的请求和响应，Request和Response，其实说白了，如果大家现在看咱们的那个自研分布式存储系统的课，里面用到了gRPC

你大概可以认为就是定义了43种接口，每个接口就是一种协议，然后每个接口都有自己对应的Request和Response，就这个意思

每个协议的Request都有相同的请求头（RequestHeader），也有不同的请求体（RequestBody），请求头包含了：api\_key、api\_version、correlation\_id、client\_id，这里的api\_key就类似于“PRODUCE”、“FETCH”，你可以认为是接口的名字吧

“PRODUCE”就是发送消息的接口，“FETCH”就是拉取消息的接口，就这个意思

api\_version，就是这个API的版本号

correlation\_id，就是类似客户端生成的一次请求的唯一标志位，唯一标识一次请求

client\_id，就是客户端的id

每个协议的Response也有相同的响应头，就是一个correlation\_id，就是对某个请求的响应

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/113\_老版本的high-level consumer的实现原理是什么？/笔记.doc**

producer和consumer api原理，都是新版本的kafka api

老版本的kafka consumer api分成两种，high-level和low-level，都是基于zk实现的，只不过前者有consumer group的概念，后者没有

high-level的api，比如说consumer启动就是在zk里写一个临时节点，但是如果自己宕机了，那么zk临时节点就没了，别人就会发现，然后就会开启rebalance

然后在消费的时候，可以指定多个线程取消费一个topic，比如说你和这个consumer分配到了5个分区，那么你可以指定最多5个线程，每个线程消费一个分区的数据，但是新版本的就一个线程负责消费所有分区

在提交offset，就是向zk写入对某个分区现在消费到了哪个offset了，默认60秒才提交一次

新版本的api就不基于zk来实现了呢，zk主要是做轻量级的分布式协调，元数据存储，并不适合高并发大量连接的场景，cnosumer可能有成百上千个，成千上万个，zk来做的，连接的压力，高并发的读写

broker内部基于zk来进行协调

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/102\_为消费者选择Coordinator的算法是如何实现的？/笔记.doc**

首先对groupId进行hash，接着对\_\_consumer\_offsets的分区数量取模，默认是50，可以通过offsets.topic.num.partitions来设置，找到你的这个consumer group的offset要提交到\_\_consumer\_offsets的哪个分区

比如说：groupId，“membership-consumer-group” -> hash值（数字）-> 对50取模 -> 就知道这个consumer group下的所有的消费者提交offset的时候是往哪个分区去提交offset，大家可以找到\_\_consumer\_offsets的一个分区

\_\_consumer\_offset的分区的副本数量默认来说1，只有一个leader

然后对这个分区找到对应的leader所在的broker，这个broker就是这个consumer group的coordinator了，接着就会维护一个Socket连接跟这个Broker进行通信

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/115\_Producer的缓冲区内部数据结构是什么样子的？/笔记.doc**

producer会创建一个accumulator缓冲区，他里面是一个HashMap数据结构，每个分区都会对应一个batch队列，因为你打包成出来的batch，那必须是这个batch都是发往同一个分区的，这样才能发送一个batch到这个分区的leader broker

{

“order-topic-0” -> [batch1, batch2],

“order-topic-1” -> [batch3]

}

batch.size

每个batch包含三个东西，一个是compressor，这是负责追加写入batch的组件；第二个是batch缓冲区，就是写入数据的地方；第三个是thunks，就是每个消息都有一个回调Callback匿名内部类的对象，对应batch里每个消息的回调函数

每次写入一条数据都对应一个Callback回调函数的

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/108\_什么情况下会导致Consumer Group之间的重平衡？/笔记.doc**

其实最主要的就是consumer所在服务比如重启了，或者宕机了，或者新部署了，此时coordinator感知到了consumer group内的变化，就会触发rebalance的操作，或者是如果动态给topic增加了分区，也会触发rebalance

也或者是这个消费组订阅了更多的topic，也会触发rebalance进行新订阅topic的分区分配给各个consumer来处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/104\_rebalance的三种策略分别有哪些优劣势？/笔记.doc**

这里有三种rebalance的策略：range、round-robin、sticky

0~8

order-topic-0

order-topic-1

order-topic-2

range策略就是按照partiton的序号范围，比如partitioin0~2给一个consumer，partition3~5给一个consumer，partition6~8给一个consumer，默认就是这个策略；

round-robin策略，就是轮询分配，比如partiton0、3、6给一个consumer，partition1、4、7给一个consumer，partition2、5、8给一个consumer

但是上述的问题就在于说，可能在rebalance的时候会导致分区被频繁的重新分配，比如说挂了一个consumer，然后就会导致partition0~4分配给第一个consumer，partition5~8分配给第二个consumer

这样的话，原本是第二个consumer消费的partition3~4就给了第一个consumer，实际上来说未必就很好

最新的一个sticky策略，就是说尽可能保证在rebalance的时候，让原本属于这个consumer的分区还是属于他们，然后把多余的分区再均匀分配过去，这样尽可能维持原来的分区分配的策略

consumer1：0~2 + 6~7

consumer2：3~5 + 8

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/121\_盘点一下在Broker内部有哪些不同场景下会有延时任务？/笔记.doc**

比如说acks=-1，那么必须等待leader和follower都写完才能返回响应，而且有一个超时时间，默认是30秒，也就是request.timeout.ms，那么在写入一条数据到leader磁盘之后，就必须有一个延时任务，到期时间是30秒

延时任务会被放到DelayedOperationPurgatory，延时操作管理器中

这个延时任务如果因为所有follower都写入副本到本地磁盘了，那么就会被自动触发苏醒，那么就可以返回响应结果给客户端了，否则的话，这个延时任务自己指定了最多是30秒到期，如果到了超时时间都没等到，那么就直接超时返回异常了

还有一种是延时拉取任务，也就是说follower往leader拉取消息的时候，如果发现是空的，那么此时会创建一个延时拉取任务，然后延时时间到了之后，就会再次读取一次消息，如果过程中leader写入了消息那么也会自动执行这个拉取任务

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/103\_Coordinator和Consume Leader如何协作制定分区方案？/笔记.doc**

每个consumer都发送JoinGroup请求到Coordinator，然后Coordinator从一个consumer group中选择一个consumer作为leader，把consumer group情况发送给这个leader，接着这个leader会负责制定分区方案，通过SyncGroup发给Coordinator

接着Coordinator就把分区方案下发给各个consumer，他们会从指定的分区的leader broker开始进行socket连接以及消费消息

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/106\_消费过程中的各种offset之间的关系是什么？/笔记.doc**

上一次提交offset，当前offset（还未提交），高水位offset，LEO

内存里记录这么几个东西：上一次提交offset，当前消费到的offset，你不断的在消费消息，不停的在拉取新的消息，不停的更新当前消费的offset，HW offset，你拉取的时候，是只能看到HW他前面的数据

LEO，leader partition已经更新到了一个offset了，但是HW在前面，你只能拉取到HW的数据，HW后面的数据，意味着不是所有的follower都写入进去了，所以不能去读取的

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/123\_Kafka的时间轮延时调度机制（二）：多层级时间轮/笔记.doc**

接着上一讲的内容，那如果这个时候来一个350毫秒之后执行的定时任务呢？已经超出当前这个时间轮的范围了，那么就放到上层时间轮，上层时间轮的tickMs就是下层时间轮的interval，也就是20ms

wheelSize是固定的，都是20，那么上层时间轮的inetrval周期就是400ms，如果再上一层的时间轮他的tickMs是400ms，那么interval周期就是8000ms，也就是8s，再上一层时间轮的tickMs是8s，interval就是160s，也就是好几分钟了，以此类推即可

反正有很多层级的时间轮，一个时间轮不够，就往上开辟一个新的时间轮出来，每个时间轮的tickMs是下级时间轮的interval，而且currentTime就跟时钟的指针一样是不停的转动的，你只要根据定时周期把他放入对应的轮子即可

每个轮子插入的时候根据currentTime，放到对应时间之后的时间格即可

比如定时350ms后执行的任务，就可以放到interval位400ms的时间轮内，currentTime自然会转动到那个时间格来执行他

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/107\_自动提交offset的语义以及导致消息丢失和重复消费的问题/笔记.doc**

默认是自动提交

auto.commit.inetrval.ms：5000，默认是5秒提交一次

如果你提交了消费到的offset之后，人家kafka broker就可以感知到了，比如你消费到了offset = 56987，下次你的consumer再次重启的时候，就会自动从kafka broker感知到说自己上一次消费到的offset = 56987

这次重启之后，就继续从offset = 56987这个位置继续往后去消费就可以了

他的语义是一旦消息给你poll到了之后，这些消息就认为处理完了，后续就可以提交了，所以这里有两种问题：

第一，消息丢失，如果你刚poll到消息，然后还没来得及处理，结果人家已经提交你的offset了，此时你如果consumer宕机，再次重启，数据丢失，因为上一次消费的那批数据其实你没处理，结果人家认为你处理了

poll到了一批数据，offset = 65510~65532，人家刚好就是到了时间提交了offset，offset = 65532这个地方已经提交给了kafka broker，接着你准备对这批数据进行消费，但是不巧的是，你刚要消费就直接宕机了

其实你消费到的数据是没处理的，但是消费offset已经提交给kafka了，下次你重启的时候，offset = 65533这个位置开始消费的，之前的一批数据就丢失了

第二，重复消费，如果你poll到消息，都处理完毕了，此时还没来得及提交offset，你的consumer就宕机了，再次重启会重新消费到这一批消息，再次处理一遍，那么就是有消息重复消费的问题

poll到了一批数据，offset = 65510~65532，你很快的处理完了，都写入数据库了，结果还没来得及提交offset就宕机了，上一次提交的offset = 65509，重启，他会再次让你消费offset = 65510~65532，一样的数据再次重复消费了一遍，写入数据库

重启kafka consumer，修改了他的代码

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/122\_Kafka的时间轮延时调度机制（一）：O(1)时间复杂度/笔记.doc**

Kafka内部有很多延时任务，没有基于JDK Timer来实现，那个插入和删除任务的时间复杂度是O(nlogn)，而是基于了自己写的时间轮来实现的，时间复杂度是O(1)，其实Netty、ZooKeeper、Quartz很多中间件都会实现时间轮

延时任务是很多很多的，大量的发送消息以及拉取消息，都会涉及到延时任务，任务数量很多，如果基于传统的JDK Timer把大量的延时任务频繁的插入和删除，时间复杂度是O(nlogn)性能比较低的

时间轮的机制，延时任务插入和删除，O(1)

简单来说，一个时间轮（TimerWheel）就是一个数组实现的存放定时任务的环形队列，数组每个元素都是一个定时任务列表（TimerTaskList），这个TimerTaskList是一个环形双向链表，链表里的每个元素都是定时任务（TimerTask）

时间轮是有很多个时间格的，一个时间格就是时间轮的时间跨度tickMs，wheelSize就是时间格的数量，时间轮的总时间跨度就是tickMs \* wheelSize（interval），然后还有一个表盘指针（currentTime），就是时间轮当前所处的时间

currentTime指向的时间格就是到期，需要执行里面的定时任务

比如说tickMs = 1ms，wheelSize = 20，那么时间轮跨度（inetrval）就是20ms，刚开始currentTime = 0，这个时候如果有一个延时2ms之后执行的任务插入进来，就会基于数组的index直接定位到时间轮底层数组的第三个元素

因为tickMs = 1ms，所以第一个元素代表的是0ms，第二个元素代表的是1ms的地方，第三个元素代表的就是2ms的地方，直接基于数组来定位就是O(1)是吧，然后到数组之后把这个任务插入其中的双向链表，这个时间复杂度也是O(1)

所以这个插入定时任务的时间复杂度就是O(1)

然后currentTime会随着时间不断的推移，1ms之后会指向第二个时间格，2ms之后会指向第三个时间格，这个时候就会执行第三个时间格里刚才插入进来要在2ms之后执行的那个任务了

这个时候如果插入进来一个8ms之后要执行的任务，那么就会放到第11个时间格上去，相比于currentTime刚好是8ms之后，对吧，就是个意思，然后如果是插入一个19ms之后执行的呢？那就会放在第二个时间格

每个插入进来的任务，他都会依据当前的currentTime来放，最后正好要让currentTime转动这么多时间之后，正好可以执行那个时间格里的任务

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/116\_消息缓冲区满的时候是阻塞住还是抛出异常？/笔记.doc**

max.block.ms，其实就是说如果写缓冲区满了，此时是阻塞住一段时间，然后什么时候抛异常，默认是60000，也就是60秒

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/114\_老版本的low-level consumer的实现原理是什么？/笔记.doc**

老版本的low-level消费者，是可以自己控制offset的，实现很底层的一些控制，但是需要自己去提交offset，还要自己找到某个分区对应的leader broker，跟他进行连接获取消息，如果leader变化了，也得自己处理，非常的麻烦

比如说storm-kafka这个插件，在storm消费kafka数据的时候，就是使用的low-level api，自己获取offset，提交写入zk中自己指定的znode中，但是在未来基本上老版本的会越来越少使用

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/110\_最新设计的rebalance分代机制可以有什么作用？/笔记.doc**

大家设想一个场景，在rebalance的时候，可能你本来消费了partition3的数据，结果有些数据消费了还没提交offset，结果此时rebalance，把partition3分配给了另外一个cnosumer了，此时你如果提交partition3的数据的offset，能行吗？

必然不行，所以每次rebalance会触发一次consumer group generation，分代，每次分代会加1，然后你提交上一个分代的offset是不行的，那个partiton可能已经不属于你了，大家全部按照新的partiton分配方案重新消费数据

consumer group generation = 1

consumer group generation = 2

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/101~124资料/117\_负责IO请求的Sender线程是如何基于缓冲区发送数据的？/笔记.doc**

Sender线程会不停的轮询缓冲区内的HashMap，看batch是否满了，或者是看linger.ms时间是不是到了，然后就得发送数据去，发送的时候会根据各个batch的目标leader broker来进行分组

因为可能不同的batch是对应不同的分区，但是不同的分区的Leader是在一个broker上的，<Node, List<ProducerBatch>>，接着会进一步封装为<Node, Request>，每个broker一次就是一个请求，但是这里可能包含很多个batch，接着就是将分组好的batch发送给leader broker，并且处理response，来反过来调用每个batch的callback函数

发送出去的Request会被放入InFlighRequests里面去保存，Map<NodeId, Deque<Request>>，这里就代表了发送出去的请求，但是还没接收到响应的

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/001\_从日处理10亿数据场景反向驱动Kafka技术讲解的思路介绍/笔记.doc**

Kafka技术，讲解的思路

按部就班的思路去讲解的话，纯讲干讲kafka自己技术本身，一块一块的讲，效果并不是太好，hdfs，在大数据平台里面主要是搭建和部署好了之后，其实平时很少是我们自己直接去使用他的

hdfs那块就核心和关键的一点，他其实是异常、性能，所以需要深入研究里面的源码，然后对源码有二次开发的能力，去解决相应的问题

平时我们很有可能是会自己直接用到kafka的，实时计算，storm，spark streaming，flink，是一个数据流处理的管道，这里无论你是用什么技术来做实时计算，你先从kafka里消费出来数据，处理，接着再把数据写回到kafka里面

hdfs，除非你是改动spark、hive、hbase底层的源码，才有可能会牵扯到你直接自己基于代码来操作hdfs，一般来说都是一些在hdfs上层的大数据系统，他们会去使用到hdfs，不是直接用hdfs，解决他的一些故障和问题

不能上来就直接kafka源码，然后讲一些源码的二次开发，跟平时日常工作中大家对kafka的使用场景就不匹配了

咱们先构造一个场景出来，模拟出来一个场景，也是基于我真实的互联网公司的从业经验，把数据量减少，简化一下，抽象出来一个模拟的场景，1000万用户的电商平台，每天会产生各种数据会有多少

每天各种数据加起来有10亿，涌入kafka里面，作为一个实时数据流，供下游的一些实时分析系统来抽取数据流分析

每秒钟会涌入多少数据，需要支撑多大的吞吐量，包括每天有多少数据需要落地在磁盘来存储，集群需要存储多大的数据量

kafka架构原理来入手，至少两周的时间，你就知道kafka整体是怎么回事了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/015\_保证写入Kafka的数据不丢失：ISR机制到底是什么意思？/笔记.doc**

光是依靠多副本机制能保证Kafka的高可用性，但是能保证数据不丢失吗？不行，因为如果leader宕机，但是leader的数据还没同步到follower上去，此时即使选举了follower作为新的leader，当时刚才的数据已经丢失了

ISR是：in-sync replica，就是跟leader partition保持同步的follower partition的数量，只有处于ISR列表中的follower才可以在leader宕机之后被选举为新的leader，因为在这个ISR列表里代表他的数据跟leader是同步的

如果要保证写入kafka的数据不丢失，首先需要保证ISR中至少有一个follower，其次就是在一条数据写入了leader partition之后，要求必须复制给ISR中所有的follower partition，才能说代表这条数据已提交，绝对不会丢失，这是Kafka给出的承诺

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/005\_Kafka在电商平台的实时数据架构中扮演了什么角色？/笔记.doc**

10亿的量

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/003\_项目背景：1000万注册用户的电商平台每天有多少次数据库更新操作？/笔记.doc**

用户行为日志，业务数据（数据库，商品、订单、库存、评价、售后，每天都会产生各种各样的变化）

1000万用户，日活300万，每天3亿用户行为日志

对于大数据实时计算来说，数据库的变更操作，增删改的操作，每次增删改操作就算一条数据，数据变更记录，需要交给大数据平台来处理

假设每天有50万个订单，商品（价格、库存、信息），订单，评价，售后，积分，促销，很多其他的数据，每天新增的都是以订单为核心的一些数据，每天都有的增量数据，其他的数据都是围绕订单来的

是在数据库新增的订单，每对应修改，对应多少个增删改数据库操作呢？5个增删改的操作，订单表，每天有250万次操作，变更记录。100张左右的表，平均每张表每天会发生的增删改的操作大概是有200万次，2亿左右的数据

每天数据库有2亿条的变更记录，就是第二种数据了，业务数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/012\_Kafka是如何通过精心设计消息格式节约磁盘空间占用开销的？/笔记.doc**

kafka的消息格式如下：

crc32，magic，attribute，时间戳，key长度，key，value长度，value

kafka是直接通过NIO的ByteBuffer以二进制的方式来保存消息的，这种二级制紧凑保存格式可以比使用Java对象保存消息要节约40%的内存空间

然后这个消息实际上是封装在一个log entry里的，你可以认为是一个日志条目吧，在kafka里认为每个partition实际上就是一个磁盘上的日志文件，写到parttion里去的消息就是一个日志，所以log entry就是一个日志

这个日志条目包含了一个offset，一个消息的大小，然后是消息自身，就是上面那个数据结构，但是这里要注意的一点，就是这个message里可能会包含多条消息压缩在一起，所以可能找一条消息，需要从这个压缩数据里遍历搜索

而且这里还有一个概念就是消息集合，一个消息集合里包含多个日志，最新名称叫做RecordBatch

后来消息格式演化为了如下所示：

（1）消息总长度

（2）属性：废弃了，已经不用

（3）时间戳增量：跟RecordBatch的时间戳的增量差值

（4）offset增量：跟RecordBatch的offset的增量差值

（5）key长度

（6）key

（7）value长度

（8）value

（9）header个数

（10）header：自定义的消息元数据，key-value对

通过时间戳、offset、key长度等都用可变长度来尽可能减少空间占用，v2版本的数据格式比v1版本的数据格式要节约很多磁盘开销

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/017\_基于ZooKeeper实现Kafka无状态可伸缩的架构设计思路/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/009\_Kafka是如何利用顺序磁盘写机制实现单机每秒几十万消息写入的？/笔记.doc**

batch微批处理，高吞吐高延迟，kafka相反，高吞吐低延迟

直接写入os的page cache中

文件，kafka仅仅是追加数据到文件末尾，磁盘顺序写，性能极高，几乎跟写内存是一样高的。磁盘随机写，你要随机在文件的某个位置修改数据，这个叫做磁盘随机写，性能是很低的，磁盘顺序写，仅仅追加数据到文件末尾

而且写磁盘的方式是顺序写，不是随机写，性能跟内存写几乎一样。就是仅仅在磁盘文件的末尾追加写，不能在文件随机位置写入

假设基于上面说的os cache写 + 磁盘顺序写，0.01毫秒，低延迟，高吞吐，每毫秒可以处理100条数据，每秒可以处理10万条数据，不需要依托类似spark straeming那种batch微批处理的机制

正是依靠了这个超高的写入性能，单物理机可以做到每秒几十万条消息写入Kafka

这种方式让kafka的写性能极高，最大程度减少了每条数据处理的时间开销，反过来就大幅度提升了每秒处理数据的吞吐量，一般kafka部署在物理机上，单机每秒写入几万到几十万条消息是没问题的

这种方式是不是就兼顾了低延迟和高吞吐两个要求，尽量把每条消息的写入性能压榨到极致，就可以实现低延迟的写入，同时对应的每秒的吞吐量自然就提升了

所以这是kafka非常核心的一个底层机制

而且这里很关键的一点，比如rabbitmq这种消息中间件，他会先把数据写入内存里，然后到了一定时候再把数据一次性从内存写入磁盘里，但是kafka不是这种机制，他收到数据直接写磁盘

只不过是写的page cache，而且是磁盘顺序写，所以写入的性能非常高，而且这样不需要让kafka自身的jvm进程占用过多内存，可以更多的把内存空间留给os的page cache来缓存磁盘文件的数据

只要能让更多的磁盘数据缓存在os cache里，那么后续消费数据从磁盘读的时候，就可以直接走os cache读数据了，性能是非常高的

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/016\_如何让Kafka集群处理请求的时候实现负载均衡的效果？/笔记.doc**

假如说很多partition的leader都在一台机器上，那么不就会导致大量的客户端都请求那一台机器？这样是不对的，kafka集群会自动实现负载均衡的算法，尽量把leader partition均匀分布在集群各个机器上

然后客户端在请求的时候，就会尽可能均匀的请求到kafka集群的每一台机器上去了，假如出现了partition leader的变动，那么客户端会感知到，然后下次就可以就可以请求最新的那个leader partition了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/002\_项目背景：1000万注册用户的电商平台每天有多少用户行为日志？/笔记.doc**

网站的用户行为日志，假设电商网站，我现在需要买一个阅读架，看书的架子

京东，我平时比较喜欢用的是京东，送货很快，自营商品，都是放在自己的仓库里，送货很快，用的比较多的是，直男，买东西都是开门见山的，女孩儿不一样，女同志，喜欢逛网站，她喜欢在网站里上下浏览，看网站推荐的一些东西

用户行为，就是说在一个网站，或者是APP，用户会做很多的行为和操作，比如说搜索一个商品，选择接筛选项来过滤筛选商品，点击一个商品进入他的详情页，加入购物车，进入购物车界面，下订单，对订单进行支付，对商品进行评价

电商网站最核心的链路就是这样的，搜索、筛选、详情页、购物车、下单、支付、评价

用户行为，每个用户每次在电商网站里都会做很多的行为，这些用户行为实际上来说会被电商网站给收集起来你的每个行为，你每次做一个行为的时候，他就会把你的这个行为做一条日志发送到后台

后台每天就会收集大量的用户行为日志

通过对用户行为日志的分析，可以让产品经理（设计网站和app）知道自己设计的网站是否受到用户的喜欢，可以让运营专员知道自己设计推广的某个促销活动他的效果如何，还可以让中高层管理人员，每天看到网站经营的数据

电商网站有1000万注册用户，那么每天会有多少人来逛这个网站呢？每天日活用户300万，平均每个人会在网站上做多少个行为呢？假设平均每个人做的行为有100次，这个算的是比较高的了。

每天就有3亿个行为，每个行为会对应一条用户行为日志，每天的用户行为日志有3亿

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/011\_Kafka的底层数据存储结构：日志文件以及offset/笔记.doc**

基本上可以认为每个partition就是一个日志文件，存在于某台Kafka服务器上，然后这个日志里写入了很多消息，每个消息在partition日志文件里都有一个序号，叫做offset，代表这个消息是日志文件里的第几条消息

但是在消费消息的时候也有一个所谓的offset，这个offset是代表消费者目前在partition日志文件里消费到了第几条消息，是两回事儿

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/006\_Kafka架构原理、生产部署、运维监控、生产消费、系统优化/笔记.doc**

kafka这个实战的课程，分为哪几个环节和步骤去讲

业务背景已经交代的比较清楚了，就是对kafka架构原理做比较深入的研究，先明白kafka的工作原理，玩儿kafak

我们就可以基于这个项目背景，来看看kafka生产环境的部署怎么来规划，每台机器的配置，参数设置，需要多少台机器，抗每天10亿数据量

对集群日常的管理、运维、监控，学习一下是怎么来做的

研究一下，如果你要往kafka里写数据，他是怎么来写的，写数据客户端的原理是什么；如果从kafka里读数据，他是怎么来读的，读数据客户端的原理是什么

就要来研究一下kafka生产环境遇到的问题：数据丢失、数据顺序、数据积压、吞吐量优化、高可用保障、生产端的优化、消费端的优化、JVM优化、性能的优化，系统优化

hdfs和kafka都有了，接下来就可以玩儿各种数据的采集了，用户行为日志，基于真实的几十张报表的需求，来反向研究，当你遇到真实的数据分析需求的时候，数据埋点应该怎么来做，用户日志怎么来采集，离线导入hdfs，实时导入kafka

数据库业务数据采集，研究canal的源码，自研一套数据同步中间件，可以把数据库的业务数据库离线导入Hdfs，实时导入kafka

爬虫，基于java自研一套分布式爬虫系统，分布式是个关键点，N多个爬虫系统协作起来分布式抓取数据，离线导入hdfs，实时导入kafka

数据采集这个大的项目就做完了，hdfs有离线数据，kafka有实时数据，而且分别有行为日志、业务数据、竞对数据，三种数据在离线和实时两种存储里都有

离线数据仓库，实时数据平台，基于真实的电商的需求来做，尽可能100张报表的真实需求来反向驱动

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/013\_如何实现TB量级的数据在Kafka集群中分布式的存储？/笔记.doc**

但是这里有一个很大的问题，就是不可能说把TB量级的数据都放在一台Kafka服务器上吧？这样肯定会遇到容量有限的问题，所以Kafka是支持分布式存储的，也就是说你的一个topic，代表了逻辑上的一个数据集

你大概可以认为一个业务上的数据集合吧，比如说用户行为日志都走一个topic，数据库里的每个表的数据分别是一个topic，订单表的增删改的变更记录进入一个topic，促销表的增删改的变更记录进入一个topic

每个topic都有很多个partition，你认为是数据分区，或者是数据分片，大概这些意思都可以，就是说这个topic假设有10TB的数据量需要存储在磁盘上，此时你给他分配了5个partition，那么每个partition都可以存放2TB的数据

然后每个partition不就可以放在一台机器上，通过这个方式就可以实现数据的分布式存储了，每台机器上都运行一个Kafka的进程，叫做Broker，以后大家记住，borker就是一个kafka进程，在一台服务器上就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/008\_为什么Spark Streaming要采取微批处理技术实现流式计算？/笔记.doc**

如果来一条数据就处理一条数据，可能会导致每条数据要处理假设1毫秒，那么此时每秒可以处理1000条数据，这就是每秒的吞吐量，但是如果采用微批处理技术呢？比如说把9毫秒内的数据收集起来一共有1000条数据，接着一次性交给引擎来处理，1毫秒就把1000条数据给处理完了。

Kafka现在采取batch思路，10毫秒处理了1000条数据，每个系统发送数据过来到处理完成花费10毫秒，延迟提高了10倍，Kafka的吞吐量提高了，每秒可以处理10万条数据，吞吐量是提升了100倍。

那么就相当于是10毫秒处理了1000条数据，每秒可以处理10万条数据，吞吐量是不是就提升了100倍？

这个就是所谓的流式计算采用的微批处理技术，你一条一条处理，每条数据都需要启动新的计算资源，有网络开销，甚至是磁盘开销。但是你一次性处理1000条，跟你一次性处理1条其实是差不多的

因为用的计算资源什么都差不多，但是在内存里一下子可以处理完1000条数据

这就是说，提升了吞吐量，但是计算的延时就增加了，一条数据过来，需要10毫秒之后才能处理完毕。但是你要是降低计算的延时，那么吞吐量就降低了，数据来了1毫秒就处理完毕，但是每秒能处理的数据量太少了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/007\_先了解两个专业技术语：什么是吞吐量和处理延迟？/笔记.doc**

研究kafka这个东西，你必须得搞清楚这两个概念，吞吐量，延迟

写数据请求发送给kafka一直到他处理成功，你认为写请求成功，假设是1毫秒，这个就说明性能很高，延迟

kafka，每毫秒可以处理1条数据，每秒可以处理1000条数据，这个单位时间内可以处理多少条数据，就叫做吞吐量，1000条数据，每条数据10kb，10mb，吞吐量相当于是每秒处理10mb的数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/004\_项目背景：电商平台每天从竞对网站抓取的同类商品数据有多少？/笔记.doc**

第三种数据，用户行为日志，还有业务库的数据，爬虫抓的竞对网站的商品数据，竞对网站每个商品的销量、价格，就可以跟自己网站的同类商品的销量和价格做一个比对，淘宝上抓数据，几十亿个商品，几亿个商品

假设有三个同类的竞争对手网站，人家每个网站的商品数量在100万左右，每天爬虫就把人家的100万个商品抓取过来就可以了，300万条数据而已，每个商品每天的销量在不停变化，爬虫可能需要不停地抓取对方网站的数据，每个商品每隔10分钟就会抓取一次，6 \* 24 = 144 = 150次

300万商品，150次，4.5亿，5亿数据

来算一下，用户行为日志（3亿），业务库变更记录（2亿），竞对数据（5亿） = 10亿

涌入到kafka里去，可以做用户行为分析，网站运营分析，竞争对手分析，协助网站的产品经理、运营人员、企业高层把控网站每天运行的方方面面，以此做出对应的一些决策

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/010\_Kafka是如何利用零拷贝和页缓存技术实现高性能读取的？/笔记.doc**

那么在消费数据的时候，需要从磁盘文件里读取数据后通过网络发送出去，这个时候怎么提升性能呢？

首先就是利用了page cache技术，之前说过，kafka写入数据到磁盘文件的时候，实际上是写入page cache的，没有直接发生磁盘IO，所以写入的数据大部分都是停留在os层的page cache里的

这个本质其实跟elasticsearch的实现原理是类似的

然后在读取的时候，如果正常情况下从磁盘读取数据，先尝试从page cache读，读不到才从磁盘IO读，读到数据以后先会放在os层的一个page cache里，接着会发生上下文切换到系统那边，把os的读缓存数据拷贝到应用缓存里

接着再次发生上下文二切换到os层，把应用缓存的数据拷贝到os的socket缓存中，最后数据再发送到网卡上

这个过程里，发生了好几次上下文切换，而且还涉及到了好几次数据拷贝，如果不考虑跟硬件之间的交互，起码是从os cache到用户缓存，从用户缓存到socket缓存，有两次拷贝是绝对没必要的

但是如果用零拷贝技术，就是linux的sendfile，就可以直接把操作交给os，os看page cache里是否有数据，如果没有就从磁盘上读取，如果有的话直接把os cache里的数据拷贝给网卡了，中间不用走那么多步骤了

对比一下，是不是所谓的零考别了？

所以呢，通过零拷贝技术来读取磁盘上的数据，还有page cahce的帮助，这个性能就非常高了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/001~018资料/014\_如何基于多副本冗余机制保证Kafka宕机时还具备高可用性？/笔记.doc**

但是这里就有一个问题了，如果此时Kafka某台机器宕机了，那么一个topic就丢失了一个partition的数据，此时不就导致数据丢失了吗？所以啊，所以对数据做多副本冗余，也就是每个parttion都有副本

比如最基本的就是每个partition做一个副本，副本放在另外一台机器上

然后呢kafka自动从一个partition的多个副本中选举出来一个leader partition，这个leader partition就负责对外提供这个partiton的数据读写，接收到写过来的数据，就可以把数据复制到副本partition上去

这个时候如果说某台机器宕机了，上面的leader partition没了，此时怎么办呢？通过zookeeper来维持跟每个kafka的会话，如果一个kafka进程宕机了，此时kafka集群就会重新选举一个leader partition，就是用他的某个副本partition即可

通过副本partition可以继续体统这个partition的数据写入和读取，这样就可以实现容错了，这个副本partition的专业术语叫做follower partition，所以每个partitino都有多个副本，其中一个是leader，是选举出来的，其他的都是follower partition

多副本冗余的机制，就可以实现Kafka高可用架构

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/020\_Partition的几个核心offset：高水位offset、LEO代表了什么？/笔记.doc**

实际上来说，每次leader接收到一条消息，都会更新自己的LEO，也就是log end offset，把最后一位offset + 1，这个大家都能理解吧？接着各个follower会从leader请求同步数据，这是持续进行的

offset = 0 ~ offset = 4，LEO = 5，代表了最后一条数据后面的offset，下一次将要写入的数据的offset，LEO，你一定要明白他的名词

然后follower同步到数据之后，就会更新自己的LEO

并不是leader主动推送数据给follower，他实际上是follower主动向leader尝试获取数据，不断的发送请求到leader来fetch最新的数据

然后对于接收到的某一条数据，所有follower的LEO都更新之后，leader才会把自己的HW（High Water Mark）高水位offset + 1，这个高水位offset表示的就是最新的一条所有follower都同步完成的消息

partition中最开始的一条数据的offset是base offset

LEO和HW分别是干什么的呢？

LEO很重要的一个功能，是负责用来更新HW的，就是如果leader和follower的LEO同步了，此时HW就可以更新

所有对于消费者来说，他只能看到base offset到HW offset之间的数据因为只有这之间的数据才表明是所有follower都同步完成的，这些数据叫做“已提交”的，也就是committed，是可以被消费到的

HW offset到LEO之间的数据，是“未提交的”，这时候消费者是看不到的

HW offset表示的是当前已经提交的数据offset，LEO表示的是下一个要写入的数据的offset

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/035\_Broker是如何基于Reactor模式进行多路复用请求处理的？/笔记.doc**

每个broker上都有一个acceptor线程和很多个processor线程，可以用num.network.threads参数设置processor线程的数量，默认是3，client跟一个broker之间只会创建一个socket长连接，他会复用

然后broker就用一个acceptor来监听每个socket连接的接入，分配这个socket连接给一个processor线程，processor线程负责处理这个socket连接，监听socket连接的数据传输以及客户端发送过来的请求，acceptor线程会不停的轮询各个processor来分配接入的socket连接

proessor需要处理多个客户端的socket连接，就是通过java nio的selector多路复用思想来实现的，用一个selector监听各个socket连接，看其是否有请求发送过来，这样一个processor就可以处理多个客户端的socket连接了

processor线程会负责把请求放入一个broker全局唯一的请求队列，默认大小是500，是queued.max.requests参数控制的，所以那几个processor会不停的把请求放入这个请求队列中

接着就是一个KafkaRequestHandler线程池负责不停的从请求队列中获取请求来处理，这个线程池大小默认是8个，由num.io.threads参数来控制，处理完请求后的响应，会放入每个processor自己的响应队列里

每个processor其实就是负责对多个socket连接不停的监听其传入的请求，放入请求队列让KafkaRequestHandler来处理，然后会监听自己的响应队列，把响应拿出来通过socket连接发送回客户端

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/026\_Kafka 0.11.x版本引入leader epoch机制解决高水位机制弊端/笔记.doc**

所谓的leader epoch大致理解为每个leader的版本号，以及自己是从哪个offset开始写数据的，类似[epoch = 0, offset = 0]，这个就是epoch是版本号的意思，接着的话，按照之前的那个故障场景

假如说follower先宕机再重启，他会找leader继续同步最新的数据，更新自己的LEO和HW，不会截断数据，因为他会看看自己这里有没有[epoch, offset]对，如果有的话，除非是自己的offset大于了leader的offset，才会截断自己的数据

而且人家leader的最新offset = 1，自己的offset = 0，明显自己落后于人家，有什么资格去截断数据呢？对不对，就是这个道理。而且还会去从leader同步最新的数据过来，此时自己跟Leader数据一致。

如果此时leader宕机，切换到follower上，此时就会更新自己的[epoch = 1, offset = 2]，意思是自己的leader版本号是epoch = 1，自己从offset = 2开始写数据的

然后接着老leader恢复变为follower，从新leader看一下epoch跟自己对比，人家offset = 2，自己的offset = 0，也不需要做任何数据截断，直接同步人家数据就可以了

然后针对数据不一致的场景，如果说老leader恢复之后作为follower，从新leader看到[epoch = 1, offset = 1]，此时会发现自己的offset也是1，但是人家新leader是从offset = 1开始写的，自己的offset = 1怎么已经有数据了呢？

此时就会截断掉自己一条数据，然后跟人家同步保持数据一致

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/024\_高水位机制可能导致leader切换时发生数据丢失问题/笔记.doc**

基于之前说的高水位机制，可能会导致一些问题，比如数据丢失

假如说生产者的min.insync.replicas设置为1，这个就会导致说生产者发送消息给leader，leader写入log成功后，生产者就会认为写成功了，此时假设生产者发送了两条数据给leader，leader写成功了

此时leader的LEO = 1，HW = 0，因为follower还没同步，HW肯定是0

接着follower发送fetch请求，此时leader发现follower LEO = 0，所以HW还是0，给follower带回去的HW也是0，然后follower开始同步数据也写入了两条数据，自己的LEO = 1，但是HW = 0，因为leader HW为0

接着follower再次发送fetch请求过来，自己的LEO = 1，leader发现自己LEO = 1，follower LEO = 1，所以HW更新为1，同时会把HW = 1带回给follower，但是此时follower还没更新HW的时候，HW还是0

这个时候假如说follower机器宕机了，重启机器之后，follower的LEO会自动被调整为0，因为会依据HW来调整LEO，而且自己的那两条数据会被从日志文件里删除，数据就没了

这个时候如果leader宕机，就会选举follower为leader，此时HW = 0，接着leader那台机器被重启后作为follower，这个follower会从leader同步HW是0，此时会截断自己的日志，删除两条数据

这种场景就会导致数据的丢失

非常极端的一个场景，数据可能会莫名其妙的丢失

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/030\_Kafka 0.9x版本之后的ISR机制做了哪些优化适应生产环境？/笔记.doc**

kafka 0.9.x之后去掉了原来的replica.lag.max.messages参数，引入了一个新的replica.lag.time.max.ms参数，默认值是10秒，这个就不按照落后的条数来判断了，而是说如果某个follower的LEO一直落后leader超过了10秒，那么才判定这个follower是out-of-sync的

这样假如说线上出现了流量洪峰，一下子导致几个follower都落后了不少数据，但是只要尽快追上来，在10秒内别一直落后，就不会认为是out-of-sync，这个机制在线上实践会发现效果要好多了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/031\_深入看看Kafka在磁盘上是如何采用分段机制保存日志的？/笔记.doc**

每个分区对应的目录，就是“topic-分区号”的格式，比如说有个topic叫做“order-topic”，那么假设他有3个分区，每个分区在一台机器上，那么3台机器上分别会有3个目录，“order-topic-0”，“order-topic-1”，“order-topic-2”

每个分区里面就是很多的log segment file，也就是日志段文件，每个分区的数据会被拆分为多个段，放在多个文件里，每个文件还有自己的索引文件，大概格式可能如下所示：

00000000000000000000.index

00000000000000000000.log

00000000000000000000.timeindex

00000000000005367851.index

00000000000005367851.log

00000000000005367851.timeindex

00000000000009936472.index

00000000000009936472.log

00000000000009936472.timeindex

这个9936472之类的数字，就是代表了这个日志段文件里包含的起始offset，也就说明这个分区里至少都写入了接近1000万条数据了

kafka broker有一个参数，log.segment.bytes，限定了每个日志段文件的大小，最大就是1GB，一个日志段文件满了，就自动开一个新的日志段文件来写入，避免单个文件过大，影响文件的读写性能，这个过程叫做log rolling

正在被写入的那个日志段文件，叫做active log segment

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/037\_如何基于Zookeeper实现Controller的选举以及故障转移/笔记.doc**

在kafka集群启动的时候，会自动选举一台broker出来承担controller的责任，然后负责管理整个集群，这个过程就是说集群中每个broker都会尝试在zk上创建一个/controller临时节点

zk的一些基础知识和临时节点是什么，百度一下zookeeper入门

但是zk会保证只有一个人可以创建成功，这个人就是所谓controller角色

一旦controller所在broker宕机了，此时临时节点消失，集群里其他broker会一直监听这个临时节点，发现临时节点消失了，就争抢再次创建临时节点，保证有一台新的broker会成为controller角色

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/021\_深入探究Leader与Follower上的LEO是如何更新的？/笔记.doc**

首先leader接收到数据字后就会更新自己的LEO值

接着follower会不断的向leader发送fetch请求同步数据，然后每次一条数据同步到follower之后，他的LEO就会更新，同时leader发送数据给follower的时候，在leader端会维护所有follower的LEO值

follower发送fetch请求给leader的时候会带上自己的LEO值，然后leader每次收到一个fetch请求就会更新自己维护的每个follower的LEO值

所以这里大家要知道的是，leader上是会保存所有follower的LEO值的，这个是非常关键和核心的一点

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/022\_深入探究Leader与Follower上的高水位offset是如何更新的？/笔记.doc**

每次leader发送数据给follower的时候，都会发送自己的HW值，然后follower获取到leader HW之后，就会跟自己的LEO比较一下，取里面小的那个值作为自己的HW值，换句话说，如果follower的LEO比leader HW大了，那么follower的HW就是leader HW

但是如果follower的LEO比leader HW小，说明自己明显落后于leader，那么follower的HW就是自己的LEO值

然后leader上的HW就很明显了，那就是主要是他在接收follower的fetch请求的时候，就会在更新自己维护的所有follower的LEO之后，判断一下当前自己的LEO是否跟所有follower都保持一致，那么就会自动更新自己的HW值

这个leader的HW值就是partition的HW值，代表了从这个partition的哪个offset之前可以被消费数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/038\_创建Topic时Kafka Controller是如何完成Leader选举的呢？/笔记.doc**

如果你现在创建一个Topic，肯定会分配几个Partition，每个partition还会指定几个副本，这个时候创建的过程中就会在zookeeper中注册对应的topic的元数据，包括他有几个partition，每个partition有几个副本，每个partition副本的状态，此时状态都是：NonExistentReplica

然后Kafka Controller本质其实是会监听zk上的数据变更的，所以此时就会感知到topic变动，接着会从zk中加载所有partition副本到内存里，把这些partition副本状态变更为：NewReplica，然后选择的第一个副本作为leader，其他都是follower，并且把他们都放到partition的ISR列表中

比如说你创建一topic，order\_topic，3个partition，每个partition有2个副本，写入zk里去

/topics/order\_topic

partitions = 3, replica\_factor = 2

[partition0\_1, partition0\_2]

[partition1\_1, partition1\_2]

[partition2\_1, partition2\_2]

从每个parititon的副本列表中取出来第一个作为leader，其他的就是follower，把这些东西给放到partition对应的ISR列表里去

每个partition的副本在哪台机器上呢？会做一个均匀的分配，把partition分散在各个机器上面，通过算法来保证，尽可能把每个leader partition均匀分配在各个机器上，读写请求流量都是打在leader partition上的

同时还会设置整个Partition的状态：OnlinePartition

接着Controller会把这个partition和副本所有的信息（包括谁是leader，谁是follower，ISR列表），都发送给所有broker让他们知晓，在kafka集群里，controller负责集群的整体控制，但是每个broker都有一份元数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/028\_Kafka为Partition维护ISR列表的底层机制是如何设计的？（二）/笔记.doc**

很多公司比较常用的一个kafka的版本，是0.8.2.x系列，这个系列的版本是非常经典的，在过去几年相当大比例的公司都是用这个版本的kafka。当然，现在很多公司开始用更新版本的kafka了，就是0.9.x或者是1.x系列的

我们先说说在0.9.x之前的版本里，这个kafka到底是如何维护ISR列表的，什么样的follower才有资格放到ISR列表里呢？

在之前的版本里，有一个核心的参数：replica.lag.max.messages。这个参数就规定了follower如果落后leader的消息数量超过了这个参数指定的数量之后，就会认为follower是out-of-sync，就会从ISR列表里移除了

咱们来举个例子好了，假设一个partition有3个副本，其中一个leader，两个follower，然后replica.lag.max.messages = 3，刚开始的时候leader和follower都有3条数据，此时HW和LEO都是offset = 2的位置，大家都同步上来了

现在来了一条数据，leader和其中一个follower都写入了，但是另外一个follower因为自身所在机器性能突然降低，导致没及时去同步数据，follower所在机器的网络负载、内存负载、磁盘负载过高，导致整体性能下降了，此时leader partition的HW还是offset = 2的位置，没动，但是LEO变成了offset = 3的位置

依托LEO来更新ISR的话，在每个follower不断的发送Fetch请求过来的时候，就会判断leader和follower的LEO相差了多少，如果差的数量超过了replica.lag.max.messages参数设置的一个阈值之后，就会把follower给踢出ISR列表

但是这个时候第二个follower的LEO就落后了leader才1个offset，还没到replica.lag.max.messages = 3，所以第二个follower实际上还在ISR列表里，只不过刚才那条消息没有算“提交的”，在HW外面，所以消费者是读不到的

而且这个时候，生产者写数据的时候，如果默认值是要求必须同步所有follower才算写成功的，可能这个时候会导致生产者一直卡在那儿，认为自己还没写成功，这个是有可能的

一共有3个副本，1个leaderr，2个是follower，此时其中一个follower落后，被ISR踢掉了，ISR里还有2个副本，此时一个leader和另外一个follower都同步成功了，此时就可以让那些卡住的生产者就可以返回，认为写数据就成功了

min.sync.replicas = 2，ack = -1，生产者要求你必须要有2个副本在isr里，才可以写，此外，必须isr里的副本全部都接受到数据，才可以算写入成功了，一旦说你的isr副本里面少于2了，其实还是可能会导致你生产数据被卡住的

假设这个时候，第二个follower fullgc持续了几百毫秒然后结束了，接着从leader同步了那条数据，此时大家LEO都一样，而且leader发现所有follower都同步了这条数据，leader就会把HW推进一位，HW变成offset = 3

这个时候，消费者就可以读到这条在HW范围内的数据了，而且生产者认为写成功了

但是要是此时follower fullgc一直持续了好几秒钟，此时其他的生产者一直在发送数据过来，leader和第一个follower的LEO又推进了2位，LEO offset = 5，但是HW还是停留在offset = 2，这个时候HW后面的数据都是消费不了的，而且HW后面的那几条数据的生产者可能都会认为写未成功

现在导致第二个follower的LEO跟leader的LEO差距超过3了，此时触发阈值，follower认为是out-of-sync，就会从ISR列表里移除了

一旦第二个follower从ISR列表里移除了，世界清静了，此时ISR列表里就leader和第一个follower两个副本了，此时leader和第一个follower的LEO都是offset = 5，是同步的，leader就会把HW推进到offset = 5，此时消费者就可以消费全部数据了，生产者也认为他们的写操作成功了

那如果第二个follower后来他的fullgc结束了，开始大力追赶leader的数据，慢慢LEO又控制在replica.lag.max.messages限定的范围内了，此时follower会重新加回到ISR列表里去

上面就是ISR的工作原理和机制，一般导致follower跟不上的情况主要就是以下三种：

（1）follower所在机器的性能变差，比如说网络负载过高，IO负载过高，CPU负载过高，机器负载过高，都可能导致机器性能变差，同步 过慢，这个时候就可能导致某个follower的LEO一直跟不上leader，就从ISR列表里移除了

我们生产环境遇到的一些问题，kafka，机器层面，某台机器磁盘坏了，物理机的磁盘有故障，写入性能特别差，此时就会导致follower，CPU负载太高了，线程间的切换太频繁了，CPU忙不过来了，网卡被其他的程序给打满了，就导致网络传输的速度特别慢

（2）follower所在的broker进程卡顿，常见的就是fullgc问题

kafka自己本身对jvm的使用是很有限的，生产集群部署的时候，他主要是接收到数据直接写本地磁盘，写入os cache，他一般不怎么在自己的内存里维护过多的数据，主要是依托os cache（缓存）来提高读和写的性能的

（3）kafka是支持动态调节副本数量的，如果动态增加了partition的副本，就会增加新的follower，此时新的follower会拼命从leader上同步数据，但是这个是需要过程的，所以此时需要等待一段时间才能跟leader同步

replica.lag.max.messages主要是解决第一种情况的，还有一个replica.lag.time.max.ms是解决第二种情况的，比如设置为500ms，那么如果在500ms内，follower没法送请求找leader来同步数据，说明他可能在fullgc，此时就会从ISR里移除

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/027\_Kafka为Partition维护ISR列表的底层机制是如何设计的？（一）/笔记.doc**

很多公司比较常用的一个kafka的版本，是0.8.2.x系列，这个系列的版本是非常经典的，在过去几年相当大比例的公司都是用这个版本的kafka。当然，现在很多公司开始用更新版本的kafka了，就是0.9.x或者是1.x系列的

我们先说说在0.9.x之前的版本里，这个kafka到底是如何维护ISR列表的，什么样的follower才有资格放到ISR列表里呢？

在之前的版本里，有一个核心的参数：replica.lag.max.messages。这个参数就规定了follower如果落后leader的消息数量超过了这个参数指定的数量之后，就会认为follower是out-of-sync，就会从ISR列表里移除了

咱们来举个例子好了，假设一个partition有3个副本，其中一个leader，两个follower，然后replica.lag.max.messages = 3，刚开始的时候leader和follower都有3条数据，此时HW和LEO都是offset = 2的位置，大家都同步上来了

现在来了一条数据，leader和其中一个follower都写入了，但是另外一个follower因为自身所在机器性能突然降低，导致没及时去同步数据，follower所在机器的网络负载、内存负载、磁盘负载过高，导致整体性能下降了，此时leader partition的HW还是offset = 2的位置，没动，但是LEO变成了offset = 3的位置

依托LEO来更新ISR的话，在每个follower不断的发送Fetch请求过来的时候，就会判断leader和follower的LEO相差了多少，如果差的数量超过了replica.lag.max.messages参数设置的一个阈值之后，就会把follower给踢出ISR列表

但是这个时候第二个follower的LEO就落后了leader才1个offset，还没到replica.lag.max.messages = 3，所以第二个follower实际上还在ISR列表里，只不过刚才那条消息没有算“提交的”，在HW外面，所以消费者是读不到的

而且这个时候，生产者写数据的时候，如果默认值是要求必须同步所有follower才算写成功的，可能这个时候会导致生产者一直卡在那儿，认为自己还没写成功，这个是有可能的

一共有3个副本，1个leaderr，2个是follower，此时其中一个follower落后，被ISR踢掉了，ISR里还有2个副本，此时一个leader和另外一个follower都同步成功了，此时就可以让那些卡住的生产者就可以返回，认为写数据就成功了

min.sync.replicas = 2，ack = -1，生产者要求你必须要有2个副本在isr里，才可以写，此外，必须isr里的副本全部都接受到数据，才可以算写入成功了，一旦说你的isr副本里面少于2了，其实还是可能会导致你生产数据被卡住的

假设这个时候，第二个follower fullgc持续了几百毫秒然后结束了，接着从leader同步了那条数据，此时大家LEO都一样，而且leader发现所有follower都同步了这条数据，leader就会把HW推进一位，HW变成offset = 3

这个时候，消费者就可以读到这条在HW范围内的数据了，而且生产者认为写成功了

但是要是此时follower fullgc一直持续了好几秒钟，此时其他的生产者一直在发送数据过来，leader和第一个follower的LEO又推进了2位，LEO offset = 5，但是HW还是停留在offset = 2，这个时候HW后面的数据都是消费不了的，而且HW后面的那几条数据的生产者可能都会认为写未成功

现在导致第二个follower的LEO跟leader的LEO差距超过3了，此时触发阈值，follower认为是out-of-sync，就会从ISR列表里移除了

一旦第二个follower从ISR列表里移除了，世界清静了，此时ISR列表里就leader和第一个follower两个副本了，此时leader和第一个follower的LEO都是offset = 5，是同步的，leader就会把HW推进到offset = 5，此时消费者就可以消费全部数据了，生产者也认为他们的写操作成功了

那如果第二个follower后来他的fullgc结束了，开始大力追赶leader的数据，慢慢LEO又控制在replica.lag.max.messages限定的范围内了，此时follower会重新加回到ISR列表里去

上面就是ISR的工作原理和机制，一般导致follower跟不上的情况主要就是以下三种：

（1）follower所在机器的性能变差，比如说网络负载过高，IO负载过高，CPU负载过高，机器负载过高，都可能导致机器性能变差，同步 过慢，这个时候就可能导致某个follower的LEO一直跟不上leader，就从ISR列表里移除了

（2）follower所在的broker进程卡顿，常见的就是fullgc问题

（3）kafka是支持动态调节副本数量的，如果动态增加了partition的副本，就会增加新的follower，此时新的follower会拼命从leader上同步数据，但是这个是需要过程的，所以此时需要等待一段时间才能跟leader同步

replica.lag.max.messages主要是解决第一种情况的，还有一个replica.lag.time.max.ms是解决第二种情况的，比如设置为500ms，那么如果在500ms内，follower没法送请求找leader来同步数据，说明他可能在fullgc，此时就会从ISR里移除

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/036\_如何对Kafka集群进行整体控制：Controller是什么东西？/笔记.doc**

不知道大家有没有思考过一个问题，就是Kafka集群中某个broker宕机之后，是谁负责感知到他的宕机，以及负责进行Leader Partition的选举？如果你在Kafka集群里新加入了一些机器，此时谁来负责把集群里的数据进行负载均衡的迁移？

包括你的kafka集群的各种元数据，比如说每台机器上有哪些partition，谁是leader，谁是follower，是谁来管理的？如果你要删除一个topic，那么背后的各种partition如何删除，是谁来控制？

还有就是比如kafka集群扩容加入一个新的broker，是谁负责监听这个broker的加入？如果某个broker崩溃了，是谁负责监听这个broker崩溃？

这里就需要一个kafka集群的总控组件，Controller。他负责管理整个kafka集群范围内的各种东西

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/025\_高水位机制可能导致leader切换时发生数据不一致问题/笔记.doc**

假设min.insync.replicas = 1，那么只要leader写入成功，生产者而就会认为写入成功

如果leader写入了两条数据，但是follower才同步了一条数据，第二条数据还没同步，假设这个时候leader HW = 2，follower HW = 1，因为follower LEO小于leader HW，所以follower HW取自己的LEO

这个时候如果leader挂掉，切换follower变成leader，此时HW = 1，就一条数据，然后生产者又发了一条数据给新leader，此时HW变为2，但是第二条数据是新的数据。接着老leader重启变为follower，这个时候发现两者的HW都是2

所以他们俩就会继续运行了

这个时候他们俩数据是不一致的，本来合理的应该是新的follower要删掉自己原来的第二条数据，跟新leader同步的，让他们俩的数据一致，但是因为依赖HW发现一样，所以就不会截断数据了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/029\_Kafka 0.8.2.x版本的ISR机制在生产环境有什么缺陷？/笔记.doc**

之前说的那套ISR机制是kafka 0.8.x系列的机制，其实是有缺陷的，那个参数默认的值是4000，也就是follower落后4000条数据就认为是out-of-sync，但是这里有一个问题，就是这个数字是固定死的

如果说现在生产端突然之间涌入几万条数据呢？是不是有可能leader瞬间刚接收到几万条消息，然后所有follower还没来得及同步过去，此时所有follower都会被踢出ISR列表？然后同步了之后，再回到ISR列表

所以这种依靠固定参数判断的机制，会导致可能在系统高峰时期，follower会频繁的踢出ISR列表再回到ISR列表，这种完全无意义的事情

一般来说在kafka 0.8.2.x系列版本上生产的时候，一遍都会把这个ISR落后 判定阈值设置的大一些，避免上述的情况出现，你可以设置个几万，10万，4000，如果你公司里没那么大的高峰并发量，每秒就是几千的并发，那就没问题了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/019\_再看ISR机制：Leader宕机时只能选举同步的Follower/笔记.doc**

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/023\_用真实场景图解剖析Leader与Follower的LEO与高水位如何更新/笔记.doc**

假设leader收到第一条数据，此时leader LEO = 1，HW = 0，因为他发现其他follower的LEO也是0，所以HW必须是0

接着follower来发送fetch请求给leader同步数据，带过去follower的LEO = 0，所以leader上维护的follower LEO = 0，更新了一下，此时发现follower的LEO还是0，所以leader的HW继续是0

接着leader发送一条数据给follower，这里带上了leader的HW = 0，因为发现leader的HW = 0，此时follower LEO更新为1，但是follower HW = 0，取leader HW

接着下次follower再次发送fetch请求给leader的时候，就会带上自己的LEO = 1，leader更新自己维护的follower LEO = 1，此时发现follower跟自己的LEO同步了，那么leader的HW更新为1

接着leader发送给follower的数据里包含了HW = 1，此时follower发现leader HW = 1，自己的LEO = 1，此时follower的HW有更新为1

5个数据：全部都要往前推进更新，需要2次请求，第一次请求是仅仅是更新两边的LEO，第二次请求是更新另外leader管理的follower LEO，以及两个HW

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/039\_删除Topic时又是如何通过Kafka Controller控制数据清理？/笔记.doc**

如果你要是删除某个Topic的话，Controller会发送请求给这个Topic所有Partition所在的broker机器，通知设置所有Partition副本的状态为：OfflineReplica，也就是让副本全部下线，接着Controller接续将全部副本状态变为：ReplicaDeletionStarted

然后Controller还要发送请求给broker，把各个partition副本的数据给删了，其实对应的就是删除磁盘上的那些文件，删除成功之后，副本状态变为：ReplicaDeletionSuccessful，接着再变为NonExistentReplica

而且还会设置分区状态为：Offline

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/034\_Kafka是如何自定义TCP之上的通信协议以及使用长连接通信的/笔记.doc**

kafka的通信主要发生于生产端和broker之间，broker和消费端之间，broker和broker之间，这些通信都是基于TCP协议进行的，大家自己看看网络课程，底层基于TCP连接和传输数据，应用层的协议，是Kafka自己自定义的

所谓自定义协议，就是定好传输数据的格式，请求格式、响应格式，这样大家就可以统一按照规定好的格式来封装、传输和解析数据了

生产端发送数据到kafka broker来，此时发送的数据是这样子的：

sent data: 一大串数据

kafka broker直接就从sent data:截取一大段数据就可以用了，如果你没有自定义一套完整的协议，是没办法进行通信的

http协议，生产端也可以发送http协议的数据给kafka broker，http请求，http响应。应用层的协议，规定了数据请求和响应的种种复杂的格式，大家全部按照这个格式和规范来走，不要乱来

request v1.1

isCache: true

一大串数据

对于生产端和broker，消费端和broker来说，还会基于TCP建立长连接（具体见网络课程），也就是维护一批长连接，然后通过固定的连接不断的传输数据，避免频繁的创建连接和销毁连接的开销

broker端会构造一个请求队列，然后不停的获取请求放入队列，后台再搞一堆的线程来获取请求进行处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/032\_引入索引文件之后如何基于二分查找快速定位数据？/笔记.doc**

日志段文件，.log文件会对应一个.index和.timeindex两个索引文件

kafka在写入日志文件的时候，同时会写索引文件，就是.index和.timeindex，一个是位移索引，一个是时间戳索引，是两种索引

默认情况下，有个参数log.index.interval.bytes限定了在日志文件写入多少数据，就要在索引文件写一条索引，默认是4KB，写4kb的数据然后在索引里写一条索引，所以索引本身是稀疏格式的索引，不是每条数据对应一条索引的

而且索引文件里的数据是按照位移和时间戳升序排序的，所以kafka在查找索引的时候，会用二分查找，时间复杂度是O(logN)，找到索引，就可以在.log文件里定位到数据了

.index

44576 物理文件（.log位置）

57976 物理文件（.log位置）

64352 物理文件（.log位置）

offset = 58892 => 57976这条数据对应的.log文件的位置

接着就可以从.log文件里的57976这条数对应的位置开始查找，去找offset = 58892这条数据在.log里的完整数据

.timeindex是时间戳索引文件，如果要查找某段时间范围内的时间，先在这个文件里二分查找找到offset，然后再去.index里根据offset二分查找找对应的.log文件里的位置，最后就去.log文件里查找对应的数据

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/019~040资料/033\_磁盘上的日志文件是按照什么策略定期清理腾出空间的？/笔记.doc**

大家可以想，不可能说每天涌入的数据都一直留存在磁盘上，本质kafka是一个流式数据的中间件，不需要跟离线存储系统一样保存全量的大数据，所以kafka是会定期清理掉数据的，这里有几个清理策略

kafka默认是保留最近7天的数据，每天都会把7天以前的数据给清理掉，包括.log、.index和.timeindex几个文件，log.retention.hours参数，可以自己设置数据要保留多少天，你可以根据自己线上的场景来判断一下

只要你的数据保留在kafka里，你随时可以通过offset的指定，随时可以从kafka楼出来几天之前的数据，数据回放一遍，下游的数据，有多么的重要，如果是特别核心的数据，在kafka这个层面，可以保留7天，甚至是15天的数据

下游的消费者消费了数据之后，数据丢失了，你需要从kafka里楼出来3天前的数据，重新来回放处理一遍

在大数据的实时分析的项目里，其实就会涉及到这个东西的一个使用，如果你今天实时分析的一些数据出错了，此时你就需要把过去几天的数据重新楼出来回放一遍，重新来算一遍。实时数据分析的结果和hadoop离线分析的结果做一个比对

你每天都会从kafka里搂出来几天前的数据，算一下，跟离线数据的结果做一个比对

kafka broker会在后台启动线程异步的进行日志清理的工作

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/067\_如何对Kafka Broker所在机器的CPU、内存、网络进行监控/笔记.doc**

一般来说如果你要对类似kafka这种集群进行监控，建议采用开源的工具，kafka-manager，kafka eagle，都可以使用，人家都做好了会通过图表的形式来展现出来你的broker所在机器的CPU、网络、磁盘、内存的一些负载

就是一个小的公司，也可以直接基于linux的命令进行监控

使用top命令就可以查看CPU的负载，使用free命令就可以查看内存的使用情况，使用iostat -d -x -k 1 5来查看磁盘的使用率

留个作业，自己去百度，linux查看磁盘负载，查看网络负载

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/061\_中大型公司如何开发可视化平台来让业务方自己管理Topic？/笔记.doc**

中小型公司里面，其实是不需要类似这样子的一个可视化平台

你们主要就是一个大数据团队，也就20个人以内，你除了kafka以外，还要负责管理和运维zookeeper、hbase，5个人是数据平台 工程师，专门负责搭建和运维各种集群，把集群给搞好，提供给负责业务的同学来使用

比如可能也就几个人搞实时这块的，flink、spark streaming之类的东西，他们会来使用kafka，如果他们需要创建或者删除topic，直接跟你说一声就可以了，他们就直接用client API去访问kafka就可以了

查看所有的topic的一些信息，可能就你平时看看就可以了，人家根本不关注这个topic他的各个分区的副本在哪些broker.id的机器上，ISR列表，都不知道ISR是什么东西，所以说这是中小公司很现实的一个场景

其实上一讲说了一些topic管理的命令，大家可以想一下，平时topic管理无非就是几块，增删查，如果有新的业务场景需要接入kafka，首先就得创建自己的topic是不是，接着有可能某一天不需要一个topic了，就得把他给删了

然后平时就是查看这个topic的列表，查询 topic的详细信息

所以说，如果这些事儿都依赖kafka管理员来做，那他也太累了吧，在中小公司里其实是可以完全依赖kafka管理员手工管理topic的，但是中大公司里，技术团队都几百人，很多不同的team都要接入kafka，完全应该提供一个自助式接入的方式

其实自助式接入也很简单，就是kafka运维团队需要做一个web平台，你用python做，用java web做，都可以，要写出来一些界面，允许用户可以自行创建topic，申请删除topic，查看topic列表，查看topic详情

比如说创建topic，你完全可以让用户填入一个表单：

（1）topic名称

（2）业务方自己预估一下这个topic每天会有多少GB的数据

（3）业务方需要这个topic的数据保留几天

接着表单发送请求到后台，你就可以调用kafka的Java API来发送请求去创建一个topic出来

如果要在java web里使用kafka api，需要引入maven依赖

<dependency>

<groupId>org.apache.kafka</groupId>

<artifactId>kafka\_2.11</artifactId>

<version>0.11.0.1</version>

</dependency>

下面的代码就是用来发送请求去创建一个topic的，在这里把用户指定的topic名称放进去，然后根据用户预估每天的数据量来计算出来需要几个Partition，通常来说一个Partition存放个10GB的数据是没问题的

所以比如用户预估一天有20GB的数据量，那么就给他2个分区就可以了

假设说一天是2GB的数据量，保留最近7天的数据，这个topic一共就是14G的数据，这个时候你给他分配一个parttiion也可以，或者给2个partition也可以，你还要根据你的机器的数量来决定

至于说副本因子，建议不要让业务方自己手动去设置，直接给一个默认的2个副本因子，提供一定的容错性就可以了

ZkUtils zk = ZkUtils.apply(“localhost:2181”, 30000, 30000, JaasUtils.ksZkSecurityEnabled());

AdminUtils.createTopic(

zk,

“test01”, 5, 2,

new Properties(),

RackAwareMode.Enforced$.MODULE$);

此外，用户可以手动指定需要保留最近几天的数据，此时可以利用API动态修改一下那个topic级别的参数，比如就保留最近一天的数据，当然前端页面可以限制一下最多只能设置保留14天的数据

Properties props = AdminUtils.fetchEntityConfig(zk, ConfigType.Topic(), “test01”);

props.put(“delete.retention.hour”, 24);

AdminUtils.changeTopicConfig(zk, “test01”, props);

接着如果用户想要删除topic，需要在web界面上提交一个申请，说明自己删除他的原因，比如某个业务不再需要这个topic里的数据了，此时由kafka管理员在后台查看到这个申请，确认没问题，再点击确定可以删除

然后后台的java api可以执行删除topic的操作：AdminUtils.deleteTopic(“zk”, “test01”)

最后给大家留个小作业，自己去探索一下AdminUtils提供的API如何查询topic列表，以及如何查看某个topic的详细信息，这些都可以展示在前端页面上，平时让业务方可以查看一下topic的信息

另外给大家说一下，大家可以想想，平时你在界面上希望看到自己的topic除了有哪些partition，还有partition的各个副本在哪些机器上，ISR列表，还希望看到的是不是就是这个topic里的消息数量？

有一个命令可以查看，大家看看：

bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic test01 --time -1，这个命令可以看到每个分区当前的最大位移

bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic test01 --time -2，这个命令可以看到每个分区当前的最小位移

把上面两个命令的最大最小位移分别相减，就知道每个分区的消息数量了，之所以要相减，是因为消息会不断的被删除，所以不能光靠最大位移判断消息数量

其实不光是脚本，他也是用的GetOffsetShell这个类，传入一些参数来做到的，大家也可以在java代码里尝试自己实现，都非常简单。现在开源的kafka运维管理的平台也有，也可以直接基于他的来创建topic等等

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/075\_如何对线上生产环境的Kafka集群进行版本升级/笔记.doc**

第一步，在所有的broker的server.properties中填写当前的版本号，比如：

inter.broker.protocol.version=0.10.0

log.message.format.version=0.10.0

第二步，在每台机器上都用新版本（比如0.10.2.0）的kafka代码替换老版本的kafka代码，然后依次重启每台机器上的broker

第三步，更新上述两个版本号为0.10.2

第四步，再次在各个机器上重启broker即可

这个过程中可能会导致消息的重复消费，因为可能某个消息没法成功的提交offset消费记录到broker，所以会导致broke给消费者重复消费某个消息

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/077\_你的公司到底要不要对Kafka集群实现安全认证与加密机制？/笔记.doc**

使用kafka，用户授权和认证，数据传输的加密，操作性的，不难，有兴趣的话，可以自己去百度一下资料，然后尝试做一下实验，留个作业

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/064\_如果某台broker机器承载了过多leader partition怎么办/笔记.doc**

现在各个业务方可以自行申请创建topic，分区数量都是自动分配和后续动态调整的，kafka本身会自动把leader partition均匀分散在各个机器上，这样可以保证每台机器的读写吞吐量都是均匀的

但是也有例外，那就是如果某些broker宕机，会导致leader partition过于集中在其他少部分几台broker上，这会导致少数几台broker的读写请求压力过高，其他宕机的broker重启之后都是follower partition，读写请求很低，造成集群负载不均衡

有一个参数，auto.leader.rebalance.enable，默认是true，每隔300秒（leader.imbalance.check.interval.seconds）会执行一次preferred leader选举，如果一台broker上的不均衡的leader超过了10%，leader.imbalance.per.broker.percentage，就会对这个broker进行选举

也可以手动执行，bin/kafka-preferred-replica-election.sh，但是不建议手动执行，让他自动执行就好了

topic创建自助的，分区扩容自动的，leader partition均匀分散也是自动的

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/062\_某个Topic的数据量太大了需要扩容partition怎么做/笔记.doc**

如果说某个topic一开始数据量很小，比如每天就几百MB的数据，那你不需要给他分配过多的分区，因为你要是搞10个分区，分散在10台机器上，每台机器就几十MB的数据，有什么意义呢

所以一般建议生产环境采取的策略是刚开始就是按照预估的数据量给合理的分区数即可，通常来说，你可以限制每个分区在几个GB到10个GB，或者最多20个GB左右，都可以，具体要看你们公司的Kafka集群的机器资源情况

那么如果后来业务不停的发展，发现topic数据量太大了，当前的几个分区所在机器有点压力了，需要扩容增加更多的topic呢？这个时候你就可以动态扩容分区了，这样就可以让新的数据慢慢分散到更多的分区，更多的机器上去

bin/kafka-topics.sh --alter --zookeeper localhost:2181 --partitions 10 --topic test-topic

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/068\_Kafka Broker有哪些核心的运行指标是值得去监控的？/笔记.doc**

下面的JMX指标，都可以基于JConsole去连接到kafka的JMX端口来监控查看，也可以基于kafka自己提供的一个工具来监控

bin/kafka-run-class.sh kafka.tools.JmxTool --object-name kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec --jmx-url service:jmx:rmi:///jndi/rmi://:9997/jmxrmi --date-format “YYYY-MM-dd HH:mm:ss” --attributes FifteenMinuteRate --reporting-interval 5000

上面的命令就是每隔5秒去监控过于15分钟的消息接收速率，kafka的JMX的指标，特别特别的多，官网去查一下，成百上千个JMX指标

（1）消息接收/发送速率：监控每个broker的负载压力

这个就是说leader broker接收消息的速率，以及follower broker接收消息的速率；还有就是leader broker发送给follower broker的速率

kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec,topic=test

（2）controller是否存活：监控controller是否存活

kafka.controller:type=KafkaController,name=ActiveControllerCount

（3）副本不足的分区数：某个分区的副本不足了

kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions

（4）leader parttion在各个broker上的数量：确认集群的各个broker负载是均衡的

kafka.server:type=ReplicaManager,name=LeaderCount

（5）ISR变化速率：监控ISR不能变化太快

kafka.server:type=ReplicaManager,name=isExpandsPerSec

kafka.server:type=ReplicaManager,name=IsrShrinksPerSec

（6）Broker IO工作线程空闲率

kafka.server:type=KafkaRequestHandlerPool,name=RequestHandlerAvgIdlePercent

（7）Broker网络处理线程空闲率

kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent

这个JMX指标是非常非常多的，大家可以自己上官网去找，平时作为一个运维人员，需要经常对核心的一些指标看一看，这样你可以对kafka集群当前的运行情况了解的很清晰，特别是在一些故障发生的时候

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/071\_通过监控对解决线上集群的资源过载、负载不均、异常问题/笔记.doc**

集群、业务、监控

（1）监控资源的使用率：对于kafka来说，就是磁盘资源的使用率，当前集群保存了多大的数据量；每台机器的吞吐量，JMX来看到，broker接收数据的速率，每秒接收多少条消息，接收多少字节的数据

如果说你本来一个集群有4台机器，总共可以放20TB的数据，随着磁盘资源的使用率的攀升，可能你们的业务在不断的增长，会可能导致你们的机器就不够了，因为马上20TB的磁盘资源就要用满了

就需要进行broker机器的扩容，多增加一些broker机器，集群有更多的磁盘空间

每个broker的吞吐量，如果最多每台机器每秒可以接收10万条消息，每秒接收500mb的数据，现在发现每台broker在高峰期的时候，吞吐量几乎都应快要打满了，此时就说明你需要扩容了

就需要多增加一些broker机器，让每个broker所在的节点承载更小的并发请求量，保证集群的健康稳定的运行

（2）集群里的负载倾斜：如果说某几台broker承载的数据量特别大，或者是承载的吞吐量特别大，此时你就应该手动进行一下数据的迁移，可以把这些broker上的partition迁移一些到负载比较轻的机器上去

（3）异常报错、JVM GC频繁：解决报错的问题，那是非常考验你的技术功底的，Kafka源码有比较深入的研究，你后续才能在他就是说有报错的时候，可以去通过源码的分析，看他到底为什么报错

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/076\_如何使用MirrorMaker工具来进行多机房Kafka集群同步/笔记.doc**

其实对很多一般公司来说，甚至说说话很多大公司来说，都不一定会做Kafka集群多机房，但是如果你要做的话，有可能是某些2B类的大数据公司，会做Kafka集群多机房，也就是在不同的机房里搞多个kafka集群

互相之间要实现数据同步，这样就实现了多机房灾备，如果一个机房都挂了，另外一个机房的kafka集群可以立马顶上去，但是你要知道这个东西的成本极高，我们还是讲一下Kafka在这里一个工具

consumer.properties

bootstrap.servers=localhost:9092

client.id=multi.datacenter.sync.consumer

group.id=multi.datacenter.sync.consumer.group

partition.assignment.strategy=org.apache.kafka.clients.consumer.RoundRobinAssignor

producer.properties

bootstrap.servers=localhost:9092

client.id=multi.datacenter.sync.producer

bin/kafka-mirror-maker.sh --consumer.config consumer.properties --producer.config producer.properties --whitelist test01

可以搞多份consumer和producer的配置，把client.id设置为不一样的，这样就可以启动多个mirror-maker来进行数据同步了，他其实就是消费某个topic的数据来写入到另外一个集群里去罢了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/069\_如何基于JMX对Kafka集群的JVM和GC进行监控？/笔记.doc**

kafka他的设计理念就是不要过度的依赖于通过jvm内存来进行数据管理，避免说过多的数据驻留在kafka jvm内存里面，会导致过多的gc

java.lang:type=GarbageCollector,name=G1 Old Generation

java.lang:type=GarbageCollector,name=G1 Young Generation

就是对垃圾回收进行监控

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/059\_在生产环境中如何正确的启动和关闭Kafka集群？/笔记.doc**

如果你是作为一个大数据平台工程师，你负责在生产环境部署一套kafka集群，选择对应的物理机，在物理机上部署kafka就可以了，接下来其实你作为一个平台工程师，你要做到五大块的事情：

（1）你需要去支持业务团队的需求：Topic的创建、管理、扩容

（2）对kafka集群进行监控：机器监控->broker监控->JVM监控，方方面面的监控

（3）日常运维和管理的：集群扩容、版本升级、管理工作

（4）从生产消息->broker处理消息->消费消息，整个链路都可能会出问题，异常报错，kafka技术大牛，精通这个技术的人，你需要能够处理kafka相关的所有的异常报错

（5）对常见一些生产技术方案，解决一些需求和问题：事务、消息幂等、顺序、0丢失、回溯、消息积压

集群如何启动，如何关闭，小的细节需要注意一下的，需要在每台机器上都依次去执行一个脚本来启动这台机器的broker，他启动之后，自然而然就会去注册自己到zk，controller就会发现这个broker

controller会负责去把最新的集群的元数据同步给所有的broker

JMX\_PORT=9997 bin/kafka-server-start.sh -daemon /server.properties

一定要用后台方式来启动kafka，这里可以在启动的时候设置好JMX端口号，这个是用来后面对Kafka的数据进行监控的

bin/kafka-server-stop.sh

这个是关闭kafka的脚本，需要使用这个脚本来关闭

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/070\_基于开源的监控工具对Kafka生产集群进行可视化监控/笔记.doc**

机器（CPU、内存、磁盘、网络），Broker（JMX指标监控），Broker JVM进程（GC JMX指标），每个broker的安装目录下都有logs目录，监控和查阅系统级的一些异常和报错，也是需要进行监控的

哪怕是你开发一个java的系统，你写好的一个系统上线在运行了，此时你也应该对这个系统的监控做到比较完善，一个是监控你部署的机器本身的负载，另外一个是在你自己的代码运行的过程中

对一些比较关键和核心的业务指标进行监控

就比如说你可以监控一下，每秒钟接口收到的订单的数量，业务指标，对这个业务指标可以进行监控，每秒最多就100笔订单，结果突然发现每秒订单数量暴增，出现了每秒钟有10000个订单

JVM的GC的一些情况，一分钟之内连续进行了3次GC，就是有问题的，你的Java系统里面肯定是有哪些地方有问题，导致fullgc特别的频繁

一旦说你的系统里任何一个地方有异常和报错，必须要报警，发短信或者邮件，都是可以的，给到你，说你的系统出现了异常，你可以去翻看那个报错的日志

留一个作业：查阅一下Kafka JMX指标，有没有哪个指标可以帮助你进行异常报错的监控，有米有其他的什么JMX的指标是比较关键的，需要你来进行监控的

----------------------------------------------------------------------------------------------

机器、业务、JVM、异常 -> 监控

当然通常来说，我们需要一个完善的工具来帮助我们监控kafka集群了，需要可视化的工具来做到这个，雅虎开源的kafka-manager是非常好的一个工具

从官网下载kafka-manager-maser的zip文件，解压缩

然后还需要先安装sbt构建工具，这个过程很慢，直接用构建好的工具

解压缩构建好的包，然后进入conf/application.conf，修改里面的kafka-manager.zkhosts，修改成你的zk地址就可以了

chmod u+x bin/kafka-manager，bin/kafka-manager，就可以运行，进入9000端口号的界面，此时要先添加一个要监控的kafka集群，但是现在只能支持到0.11.0.0，我们部署的kafka版本太新了，还监控不了

所以大家最好还是在生产环境考虑一下，是用最新版本的，还是用老一点点的0.11.0.0版本

自己注意勾选一个JMX指标轮询就可以，然后可以实时看到JMX指标了，而且包括机器本身的指标，葛洪东西，你都是可以看到的，还有就是当前的consumer group等，你有哪些消费者，都可以看到

这不是什么很难的事情

如果你发现开源的工具无法完全满足你的监控的需求，很正常，监控类的工具开发起来是很容易的，你需要的东西，人家kafka都通过API，或者一些脚本，提供给你了，你完全可以自己也封装一个监控的平台，把我说的那些监控全部都给实现了

机器、核心的JMX指标、JVM GC、异常日志监控，报警某块，你可以配置一些报警的阈值

比如说，broker接收数据的速率如果超过了300mb/s，就要进行报警，完全可以跟一些第三方的短信和邮件的平台集成起来，一旦说有报警，就发送短信或者邮件到对应的同事的手机上和邮箱里去

一开始可能最多就投入个1人/月，1个数据工程师，1个月的时间，就可以把这个kafka监控报警平台可以做出来了，后续就是这个人抽一点点的时间维护就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/060\_作为Kafka集群管理员如何对根据业务需求管理Topic？/笔记.doc**

手头已经有了一套生产集群了，都部署好了，压力测试过后集群整体抗个每秒几十万的请求都可以做到了，支撑几十TB的数据量都可以了

平时作为kafka平台工程师，你需要协助业务团队来维护topic，各个业务团队，小公司，主要就是一个后端技术团队，或者是大数据技术团队，你自己本身也是大数据技术团队里面的一个人，只不过你的leader专门让你来负责维护kafka集群

比如说你们团队里，有的哥儿们是做实时计算这块的，他需要把实时的数据流引入到kafka里去，接着他会需要去使用flink、spark streaming、storm，这种实时计算技术，来从kafka里消费数据，接着继续去运算，做实时的一些分析

风控技术团队，推荐技术团队，他们可能也需要将一些实时的数据流引入到kafka，或者是人家直接从你这里获取实时的数据流，进行风控，或者是实时的个性化推荐，可能性都有，所以他们就是你的业务方

bin/kafka-topics.sh --create --zookeeper localhost:2181 --partitions 6 --replication-factor 2 --topic test01

创建topic其实主要是指定这个topic的分区数量，具体来说你要综合考虑Kafka集群的整体规模，有几台机器，还有就是这个topic最近7天的数据量会有多大。举个例子，某个topic每天要放用户行为日志

每天大概是1GB的数据量，最近7天就需要保留7GB的数据量，还有两个副本因子，那么就是一共14GB的数据量。现在假设Kafka集群有4台物理机，你觉得应该给这个Topic分配多少个分区？

很明显了，就4个分区就足够了，每台机器会放一个leader partition，然后都有一个follower partition，这样这个topic的数据会均匀分散在4台机器上，而且对这个topic的读写请求都均匀分散在4台机器上了

假设说你对这个topic每秒写入并发是10万条，4个分区，写入的时候会写入4个分区，在4台机器上有leader partition，均匀的分散在4台机器上，每台机器其实就是大概每秒写入2.5万个请求

那如果你的Kafka物理集群有20台机器呢？这个时候初始分区数量可以设置为7，把7个分区分散在其中的7台机器上，这样的好处，就是对这个Topic的读写请求会均匀分散在7台机器上上，对Topic的读写吞吐量是不是会更高？

所以在创建Topic的时候，大家一定要考虑好这一点，至于说副本因子的话，其实一般就是2就足够了，因为双副本可以保证一定的数据容错性，哪怕一台机器宕机了，还有其他副本保证数据不丢失，可以继续使用

如果你设置为3副本，那当然更好了，最多允许2台机器宕机不会丢数据，但是问题是你的数据存储空间会triple一下，那你的机器资源就会耗费更多了，所以这里要权衡一下。如果希望数据严格不丢失

那么min.insync.replicas，那个参数可以设置为2，要求比如ISR里有2个副本，才能写入成功，acks=-1，就是必须写入2个副本才行，这样可以保证写成功了一条数据，绝对一般是数据不会丢的

对于大数据的场景，要求的是吞吐量，而不是数据不丢失

但是如果要求更高的吞吐量，那么就设置min.insync.replicas，这个参数就是设置为1就可以了，只要有1个副本就可以写入，写入1个副本就算写成功，也就是写入leader就够了，可能数据会丢失，但是吞吐量很高，因为写入不需要等待副本同步成功

通常情况下建议采取高吞吐的策略，不要为了数据0丢失牺牲掉吞吐量，除非是极为核心的业务数据，比如说广告计费的数据

bin/kafka-topics.sh --delete --zookeeper localhost:2181 --topic test01，可以删除topic

如果要删除topic，需要设置delete.topic.enable为true，允许用户删除topic，删除之后是后台异步执行的，要过很长时间才能删除掉topic

bin/kafka-topics.sh --zookeeper localhost:2181 --list，可以查看topic列表，通过这个命令可以快速查看当前有哪些topic，还有就是可以看看要删除的topic删除掉了没有，平时这个命令管理员用的是很多的

bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic test01，可以查看topic详情，这个命令其实相当的实用，因为平时肯定要经常看看一个topic的具体情况，每个partition的leader在哪台机器上，副本在哪些机器上，ISR列表

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/063\_如何基于自研平台监控分区数据量增长以及自动化扩容分区？/笔记.doc**

小公司，20个人的数据团队，其中就几个人写的实时的应用会来使用kafka，那么其实还是真的还好，你平时作为一个kafka的管理员，多看看topic的数据增长，很多团队都接入了kafka，在里面创建了成百上千个topic

你的每一个topic的初始分区的数量是通过人家填写的表单来估算的，每天有多少GB的数据，你就可以预估一下大概需要几个分区，10GB以内的数据，可以刚开始就给他一个分区也可以了

其实通过kafka管理员手动执行命令扩容topic分区数量，是可以的，但是仅限于中小公司，因为业务方不多，没那么复杂，但是如果在中大型公司，那么就是不可能的了，此时就需要有一个平台可以全自动的去监控各个topic的分区数据量的增长，以及自动进行分区数量的扩容

其实也很简单的，你在你的kafka运维管理平台里写一个后台自己每天凌晨运行的程序，他自己每天去扫描一下每个topic的数据量，跟最开始的那个数据量进行比对，比如一个topic刚开始上线一天是10GB，但是后来慢慢发现变成一天有20GB了

那么此时，你就可以程序自动扩容，自动调用kafka api来修改那个topic的分区数，提升一倍即可，这个api给大家留个作业自己去找，自己动手写一下代码，其实非常非常的简单，自己百度百度，查查官方文档

一开始某个topic每天就3GB的数据，一开始给了一个分区

后来呢你每天都通过API扫描这个topic的数据量，后来慢慢的发现这个topic的数据量怎么变成了6GB，业务在增长，自动后台调用一个API提高这个topic的分区数量，变成2了，利用两台机器的资源来存放这个topic了

稍微花点时间，封装一个好用的kafka运维管理平台，人家自助式接入创建topic，还可以自动监控topic数据量的增长，自动进行topic的分区扩容，对业务的支撑，全部都实现web平台的自助式，自动化

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/065\_对核心业务数据Topic需要动态增加副本因子怎么做？/笔记.doc**

如果对核心业务数据需要增加副本因子

{“version”:1,”partitions”:[{“topic”:”test01”,”partition”:0,”replicas”:[0,1,2]},{“topic”:”test01”,”partition”:1,”replicas”:[0,1,2]},{“topic”:”test01”,”partition”:2,”replicas”:[0,1,2]}]}

bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file increase-replication-factor.json --execute

bin/kafka-reassign-partitions.sh --zookeeper --reassignment-json-file increase-replication-factor.json --verify

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/072\_如何对Kafka集群进行动态扩容加入更多的Broker节点？/笔记.doc**

之前大家了解过原理了，现在都知道，其实扩容增加broker是很容易的，只要配置好这个broker，然后穷，他会自动在zk里加入自己的节点，然后这个时候Controller会感知到他的加入，同步给其他所有的broker

而且Controller还会自动把集群元数据同步给这个新的broker，就是有哪些topic，每个topic有多少partitioin，每个partition有几个副本，以及对应的ISR

但是刚启动的broker不会自动被负载均衡迁移一些partiton，需要手动进行partiton在集群里的负载均衡，这样让每台broker机器管理均匀的数据量，同时分配leader partition承载均匀的读写请求流量

比如说现在你有3个topic，每个topic有5个partition，每个partitino有2个副本，一共有partition副本数量有30个，比如原来有3台broker，每台broker有10个partition副本，现在假如了第四台broker了

partition概念，资源都集中在partition里面了，数据是在partition作为单位来存放的，读写负载也是针对leader partition来进行的

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/078\_最后总结一下你应该如何管理线上生产Kafka集群/笔记.doc**

核心原理 -> 集群部署 -> 支持业务 -> 集群监控 -> 运维管理 -> 故障处理（最有技术含量）-> Kafka平台工程师 -> 数据开发工程师 -> 生产数据 -> 消费数据 -> 生产方案

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/073\_Kafka集群扩容之后如何迁移partition保证机器负载均衡？/笔记.doc**

vi topics-to-move.json

{“topics”: [{“topic”: “test01”}, {“topic”: “test02”}], “version”: 1} // 把你所有的topic都写在这里

bin/kafka-reassgin-partitions.sh --zookeeper hadoop03:2181,hadoop04:2181,hadoop05:2181 --topics-to-move-json-file topics-to-move.json --broker-list “5,6” --generate // 把你所有的包括新加入的broker机器都写在这里，就会说是把所有的partition均匀的分散在各个broker上，包括新进来的broker

此时会生成一个迁移方案，可以保存到一个文件里去：expand-cluster-reassignment.json

bin/kafka-reassign-partitions.sh --zookeeper hadoop01:2181,hadoop02:2181,hadoop03:2181 --reassignment-json-file expand-cluster-reassignment.json --execute

bin/kafka-reassign-partitions.sh --zookeeper hadoop01:2181,hadoop02:2181,hadoop03:2181 --reassignment-json-file expand-cluster-reassignment.json --verify

这种数据迁移操作一定要在晚上低峰的时候来做，因为他会在机器之间迁移数据，非常的占用带宽资源

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/066\_应该如何对你的Kafka集群设计生产级监控方案？/笔记.doc**

如何支撑业务这块，基本上你就已经思路比较清晰了，其实接下来这个业务方就会大量的写入数据到kafka以及从kafka消费数据出去，对你来说，你是kafka数据平台的工程师，你需要干的事情就是监控好这个集群

作为一个kafka管理员，在部署好了生产集群之后，要干的一个非常核心的事情，就是对集群设计一整套监控方案，基于开源工具完成部署，每天都对集群进行监控，所以这里就要先搞明白需要监控哪些东西

（1）Broker集群的运行状态：包括每个broker上管理了多少个partition，有多大的数据量，磁盘使用情况，物理机的CPU、内存、磁盘、网络的监控，比如CPU负载，内存使用率，磁盘IO负载，网络IO负载，JVM GC的情况

（2）ZooKeeper集群的运行状态：同理，跟上面是类似的，主要是磁盘使用率，然后物理机的CPU、内存、磁盘、网络的监控，这个东西一般来说可以先排除在外，可能不是你同时在维护的zk集群

（3）Topic的状态：这个你需要每天知道集群里有多少Topic，每个Topic有多少分区，有多大数据量，每个分区的副本在哪些机器上，ISR列表，等等

（4）后台定是运行的作业：比如说partition负载均衡自动迁移数据，Leader partition的重新选举，等等

（5）Kafka、生产、消费、集群是否有异常报错日志

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/059~078资料/074\_对负载不均的问题通过手工迁移分区实现集群均衡/笔记.doc**

vi topics-to-move.json

{“topics”: [{“topic”: “test01”}, {“topic”: “test02”}], “version”: 1} // 把你所有的topic都写在这里

bin/kafka-reassgin-partitions.sh --zookeeper hadoop03:2181,hadoop04:2181,hadoop05:2181 --topics-to-move-json-file topics-to-move.json --broker-list “5,6” --generate // 把你所有的包括新加入的broker机器都写在这里，就会说是把所有的partition均匀的分散在各个broker上，包括新进来的broker

此时会生成一个迁移方案，可以保存到一个文件里去：expand-cluster-reassignment.json

bin/kafka-reassign-partitions.sh --zookeeper hadoop01:2181,hadoop02:2181,hadoop03:2181 --reassignment-json-file expand-cluster-reassignment.json --execute

bin/kafka-reassign-partitions.sh --zookeeper hadoop01:2181,hadoop02:2181,hadoop03:2181 --reassignment-json-file expand-cluster-reassignment.json --verify

这种数据迁移操作一定要在晚上低峰的时候来做，因为他会在机器之间迁移数据，非常的占用带宽资源

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/082\_发送消息给Broker时遇到的各种异常该如何处理？/笔记.doc**

之前我们看到不管是异步还是同步，都可能让你处理异常，常见的异常如下：

LeaderNotAvailableException：这个就是如果某台机器挂了，此时leader副本不可用，会导致你写入失败，要等待其他follower副本切换为leader副本之后，才能继续写入，此时可以重试发送即可

如果说你平时重启kafka的broker进程，肯定会导致leader切换，一定会导致你写入报错，是LeaderNotAvailableException

NotControllerException：这个也是同理，如果说Controller所在Broker挂了，那么此时会有问题，需要等待Controller重新选举，此时也是一样就是重试即可

NetworkException：网络异常，重试即可

我们之前配置了一个参数，retries，他会自动重试的，但是如果重试几次之后还是不行，就会提供Exception给我们来处理了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/084\_消息批量发送的核心参数batch.size是如何优化吞吐量？/笔记.doc**

batch.size，设置meigebatch的大小，如果batch太小，会导致频繁网络请求，吞吐量下降；如果batch太大，会导致一条消息需要等待很久才能被发送出去，而且会让内存缓冲区有很大压力，过多数据缓冲在内存里

默认值是：16384，就是16kb，也就是一个batch满了16kb就发送出去，一般在实际生产环境，这个batch的值可以增大一些来提升吞吐量，可以自己压测一下

还有一个参数，linger.ms，这个值默认是0，意思就是消息必须立即被发送，但是这是不对的，一般设置一个100毫秒之类的，这样的话就是说，这个消息被发送出去后进入一个batch，如果100毫秒内，这个batch满了16kb，自然就会发送出去

但是如果100毫秒内，batch没满，那么也必须把消息发送出去了，不能让消息的发送延迟时间太长，也避免给内存造成过大的一个压力

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/095\_消费者offset的记录方式以及基于内部topic的提交模式/笔记.doc**

每个consumer内存里数据结构保存对每个topic的每个分区的消费offset，定期会提交offset，老版本是写入zk，但是那样高并发请求zk是不合理的架构设计，zk是做分布式系统的协调的，轻量级的元数据存储，不能负责高并发读写，作为数据存储

所以后来就是提交offset发送给内部topic：\_\_consumer\_offsets，提交过去的时候，key是group.id+topic+分区号，value就是当前offset的值，每隔一段时间，kafka内部会对这个topic进行compact

也就是每个group.id+topic+分区号就保留最新的那条数据即可

而且因为这个\_\_consumer\_offsets可能会接收高并发的请求，所以默认分区50个，这样如果你的kafka部署了一个大的集群，比如有50台机器，就可以用50台机器来抗offset提交的请求压力，就好很多

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/081\_基于Java API写一个Kafka Producer发送消息的代码示例/笔记.doc**

Properties props = new Properties();

// 这里可以配置几台broker即可，他会自动从broker去拉取元数据进行缓存

props.put(“bootstrap.servers”, “localhost:9092”);

// 这个就是负责把发送的key从字符串序列化为字节数组

props.put(“key.serializer”, “org.apache.kafka.common.serialization.StringSerializer”);

// 这个就是负责把你发送的实际的message从字符串序列化为字节数组

props.put(“value.serializer”, “org.apache.kafka.common.serialization.StringSerializer”);

props.put(“acks”, “-1”);

props.put(“retries”, 3);

props.put(“batch.size”, 323840);

props.put(“linger.ms”, 10);

props.put(“buffer.memory”, 33554432);

props.put(“max.block.ms”, 3000);

Producer<String, String> producer = new KafkaProducer<String, String>(props);

// 这是异步发送的模式

ProducerRecord<String, String> record = new ProducerRecord<>(“my-topic”, “key”, “value”);

producer.send(record, new Callback() {

@Override

public void onCompetion(RecordMetadata metadata, Exception exception) {

if(exception == null) {

// 消息发送成功

} else {

// 消息发送失败，需要重新发送

}

}

});

// 这是同步发送的模式

producer.send(record).get();

producer.close();

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/085\_如何根据业务场景对消息大小以及请求超时进行合理的设置？/笔记.doc**

max.request.size：这个参数用来控制发送出去的消息的大小，默认是1048576字节，也就1mb，这个一般太小了，很多消息可能都会超过1mb的大小，所以需要自己优化调整，把他设置更大一些

你发送出去的一条大数据，超大的JSON串，超过1MB，就不让你发了

request.timeout.ms：这个就是说发送一个请求出去之后，他有一个超时的时间限制，默认是30秒，如果30秒都收不到响应，那么就会认为异常，会抛出一个TimeoutException来让我们进行处理

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/080\_用一张图告诉你Producer发送消息的内部实现原理/笔记.doc**

每次发送消息都必须先把数据封装成一个ProducerRecord对象，里面包含了要发送的topic，具体在哪个分区，分区key，消息内容，timestamp时间戳，然后这个对象交给序列化器，变成自定义协议格式的数据

接着把数据交给partitioner分区器，对这个数据选择合适的分区，默认就轮询所有分区，或者根据key来hash路由到某个分区，这个topic的分区信息，都是在客户端会有缓存的，当然会提前跟broker去获取

接着这个数据会被发送到producer内部的一块缓冲区里

然后producer内部有一个Sender线程，会从缓冲区里提取消息封装成一个一个的batch，然后每个batch发送给分区的leader副本所在的broker

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/098\_对消息进行消费时有哪几个参数需要注意以及设置呢？/笔记.doc**

fetch.max.bytes：获取一条消息最大的字节数，一般建议设置大一些

max.poll.records：一次poll返回消息的最大条数，默认是500条

connection.max.idle.ms：consumer跟broker的socket连接如果空闲超过了一定的时间，此时就会自动回收连接，但是下次消费就要重新建立socket连接，这个建议设置为-1，不要去回收

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/090\_Kafka Producer高阶用法（一）：自定义分区/笔记.doc**

public class HotDataPartitioner implements Partitioner {

private Random random;

@Override

public void configure(Map<String, ?> configs) {

random = new Random();

}

@Override

public int partition(String topic, Object keyObj, byte[] keyBytes, Object value, byte[] valueBytes, Cluster cluster) {

String key = (String)keyObj;

List<PartitionInfo> partitionInfoList = cluster.availablePartitionsForTopic(topic);

int partitionCount = partitionInfoList.size();

int hotDataPartition = partitionCount - 1;

return !key.contains(“hot\_data”) ? random.nextInt(partitionCount - 1) : hotDataPartition;

}

}

props.put(“partitioner.class”, “com.zhss.HotDataPartitioner”);

测试发送

bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic test-topic

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/087\_针对瞬间异常的消息重试参数有哪些需要考虑的点/笔记.doc**

有的时候一些leader切换之类的问题，需要进行重试，设置retries即可，而且还可以跟消息不丢失结合起来，但是消息重试会导致重复发送的问题，比如说网络抖动一下导致他以为没成功，就重试了，其实人家都成功了

所以消息重试导致的消费重复，需要你在下游consumer做幂等性处理，但是kafka已经支持了一次且仅一次的消息语义

另外一个，消息重试是可能导致消息的乱序的，因为可能排在你后面的消息都发送出去了，你现在收到回调失败了才在重试，此时消息就会乱序，所以可以使用“max.in.flight.requests.per.connection”参数设置为1，这样可以保证producer同一时间只能发送一条消息

两次重试的间隔默认是100毫秒，用“retry.backoff.ms”来进行设置

一般来说，某台broker重启导致的leader切换，是最常见的异常，所以尽可能把重试次数和间隔，设置的可以cover住新leader切换过来

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/099\_消费者offset相关的参数设置会对运行产生什么样的影响？/笔记.doc**

auto.offset.reset：这个参数的意思是，如果下次重启，发现要消费的offset不在分区的范围内，就会重头开始消费；但是如果正常情况下会接着上次的offset继续消费的

enable.auto.commit：这个就是开启自动提交唯一

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/083\_发送消息的缓冲区应该如何优化来提升发送的吞吐量？/笔记.doc**

buffer.memory：设置发送消息的缓冲区，默认值是33554432，就是32MB

如果发送消息出去的速度小于写入消息进去的速度，就会导致缓冲区写满，此时生产消息就会阻塞住，所以说这里就应该多做一些压测，尽可能保证说这块缓冲区不会被写满导致生产行为被阻塞住

compression.type，默认是none，不压缩，但是也可以使用lz4压缩，效率还是不错的，压缩之后可以减小数据量，提升吞吐量，但是会加大producer端的cpu开销

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/096\_基于Java API写一个Kafka Consumer消费消息的代码示例/笔记.doc**

String topicName = “test-topic”;

String groupId = “test-group”;

Properties props = new Properties();

props.put(“bootstrap.servers”, “localhost:9092”);

props.put(“group.id”, “groupId”);

props.put(“enable.auto.commit”, “true”);

props.put(“auto.commit.ineterval.ms”, “1000”);

// 每次重启都是从最早的offset开始读取，不是接着上一次

props.put(“auto.offset.reset”, “earliest”);

props.put(“key.deserializer”, “org.apache.kafka.common.serialization.StringDeserializer”);

props.put(“value.deserializer”, “org.apache.kafka.common.serialization.SttringDeserializer”);

KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(props);

consumer.subscribe(Arrays.asList(topicName));

try {

while(true) {

ConsumerRecords<String, String> records = consumer.poll(1000); // 超时时间

for(ConsumerRecord<String, String> record : records) {

System.out.println(record.offset() + “, ” + record.key() + “, ” + record.value());

}

}

} catch(Exception e) {

}

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/093\_案例实战：电商平台的订单支付以及会员积分（三）/笔记.doc**

你的生产者，比如说订单系统，可以部署在很多台机器上，每个机器部署的服务实例，JVM进程，KafkaProducer实例，非常重的东西，线程资源，Sender线程，内存资源，缓冲区，网络资源，维护Socket

每个生产者，都跟每个Broker都维护一个Socket连接

其实工作原理都是一样的，分区路由，把你的消息路由到对应的分区里面去，往那个分区的leader broker去发送消息过去

消费者，会员系统，需要从kafka里消费数据，部署很多台机器，每台机器一定是有一个KafkaConsumer对象实例，内部肯定也会维护一些网络资源，内存资源，线程资源，会员系统实例内部都有一个KafkaConsumer对象实例

部署无限多的机器都可以正常的运作呢？各个KafkaConsumer到底是如何从各个分区里拉取数据的呢？

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/079\_Kafka Producer怎么把消息发送给Broker集群的？/笔记.doc**

需要指定把消息发送到哪个topic去

首先需要选择一个topic的分区，默认是轮询来负载均衡，但是如果指定了一个分区key，那么根据这个key的hash值来分发到指定的分区，这样可以让相同的key分发到同一个分区里去，还可以自定义partitioner来实现分区策略

producer.send(msg); // 用类似这样的方式去发送消息，就会把消息给你均匀的分布到各个分区上去

producer.send(key, msg); // 订单id，或者是用户id，他会根据这个key的hash值去分发到某个分区上去，他可以保证相同的key会路由分发到同一个分区上去

知道要发送到哪个分区之后，还得找到这个分区的leader副本所在的机器，然后跟那个机器上的Broker通过Socket建立连接来进行通信，发送Kafka自定义协议格式的请求过去，把消息就带过去了

如果找到了partition的leader所在的broker之后，就可以通过socket跟那台broker建立连接，接着发送消息过去

Producer（生产者客户端），起码要知道两个元数据，每个topic有几个分区，每个分区的leader是在哪台broker上，会自己从broker上拉取kafka集群的元数据，缓存在自己client本地客户端上

kafka核心原理、集群部署、运维管理 -> 初步玩儿起来一套kafka集群

kafka使用者的层面来考虑一下，我如果要把数据写入kafka集群，应该如何来做，怎么把数据写入kafka集群，以及他背后的一些原理还有使用过程中需要设置的一些参数，到底应该怎么来弄

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/097\_Kafka感知消费者故障是通过哪三个参数来实现的？/笔记.doc**

heartbeat.interval.ms：consumer心跳时间，必须得保持心跳才能知道consumer是否故障了，然后如果故障之后，就会通过心跳下发rebalance的指令给其他的consumer通知他们进行rebalance的操作

session.timeout.ms：kafka多长时间感知不到一个consumer就认为他故障了，默认是10秒

max.poll.interval.ms：如果在两次poll操作之间，超过了这个时间，那么就会认为这个consume处理能力太弱了，会被踢出消费组，分区分配给别人去消费，一遍来说结合你自己的业务处理的性能来设置就可以了

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/094\_一张图画清Kafka基于Consumer Group的消费者组的模型/笔记.doc**

每个consumer都要属于一个consumer.group，就是一个消费组，topic的一个分区只会分配给一个消费组下的一个consumer来处理，每个consumer可能会分配多个分区，也有可能某个consumer没有分配到任何分区

分区内的数据是保证顺序性的

group.id = “membership-consumer-group”

如果你希望实现一个广播的效果，你的每台机器都要消费到所有的数据，每台机器启动的时候，group.id可以是一个随机生成的UUID也可以，你只要让不同的机器的KafkaConsumer的group.id是不一样的

如果consumer group中某个消费者挂了，此时会自动把分配给他的分区交给其他的消费者，如果他又重启了，那么又会把一些分区重新交还给他，这个就是所谓的消费者rebalance的过程

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/089\_案例实战：电商平台的订单支付以及会员积分（二）/笔记.doc**

订单系统一旦发现一个订单支付成功了，就会发送一条消息给会员系统，kafka中转，就会根据金额累加对应的积分，假设支付成功1块钱，就累加1积分；如果订单支付成功了，后面又申请退款了，订单系统发现订单退款成功了，此时又要发送消息到kafka，会员系统发现订单退款了，此时扣减积分

orderId，如果说你订单瞬间，一下子支付成功，然后立马就退款，同一个orderId，先是支付接着是退款，两条操作信息一起发出去两条消息，一扣减积分，突然发现可能用户会看到说积分是负数

必须是先获取到支付订单的消息，然后再获取到订单退款的消息，先增加积分，再扣减积分，按照顺序来走的

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/086\_基于Kafka内核架构原理深入分析acks参数到底是干嘛的/笔记.doc**

acks参数，其实是控制发送出去的消息的持久化机制的

如果acks=0，那么producer根本不管写入broker的消息到底成功没有，发送一条消息出去，立马就可以发送下一条消息，这是吞吐量最高的方式，但是可能消息都丢失了，你也不知道的，但是说实话，你如果真是那种实时数据流分析的业务和场景，就是仅仅分析一些数据报表，丢几条数据影响不大的

会让你的发送吞吐量会提升很多，你发送弄一个batch出，不需要等待人家leader写成功，直接就可以发送下一个batch了，吞吐量很大的，哪怕是偶尔丢一点点数据，实时报表，折线图，饼图

acks=all，或者acks=-1：这个leader写入成功以后，必须等待其他ISR中的副本都写入成功，才可以返回响应说这条消息写入成功了，此时你会收到一个回调通知

min.insync.replicas = 2，ISR里必须有2个副本，一个leader和一个follower，最最起码的一个，不能只有一个leader存活，连一个follower都没有了

acks = -1，每次写成功一定是leader和follower都成功才可以算做成功，leader挂了，follower上是一定有这条数据，不会丢失

retries = Integer.MAX\_VALUE，无限重试，如果上述两个条件不满足，写入一直失败，就会无限次重试，保证说数据必须成功的发送给两个副本，如果做不到，就不停的重试，除非是面向金融级的场景，面向企业大客户，或者是广告计费，跟钱的计算相关的场景下，才会通过严格配置保证数据绝对不丢失

acks=1：只要leader写入成功，就认为消息成功了，默认给这个其实就比较合适的，还是可能会导致数据丢失的，如果刚写入leader，leader就挂了，此时数据必然丢了，其他的follower没收到数据副本，变成leader

# **File Path: /Users/xiaotingting/Downloads/学习资料/03\_支撑每日10亿数据量的Kafka架构实战unzip/079~100资料/088\_案例实战：电商平台的订单支付以及会员积分（一）/笔记.doc**

小小的案例，电商平台里，比如说你要是下了一个订单，人家一般来说会让你来支付，支付了一个订单之后，是不是比如说会员系统需要获取到新支付订单的情况，这样就可以根据订单支付的金额，给你累加对应的积分

订单系统，他如果有支付成功的订单之后，就发送到kafka里去，他作为一个kafka producer，生产者，会员系统消费到这个订单之后，就可以根据这个订单对应的金额去累加人家对应的积分了

订单和会员，生产者，消费者，做一些小的案例，实践一些技术，帮助大家更好的理解他里面的一些用法