# MQ

## 你们公司生产环境用的是什么消息中间件？

Kafka,虽然未参与过选型，不过大概知道为什么。对比于ActiveMQ，RabbitMQ，RocketMQ，Kafka的优势在于专为超高吞吐量的实时日志采集、实时数据同步、实时数据计算等场景来设计。

## 为什么在你们系统架构中要引入消息中间件？

复杂系统的解耦

复杂链路的异步调用（将调用链路中操作比较耗时的模块抽出来，通过MQ调用）

瞬时高峰的削峰处理

分布式事物

## 如何保证消息中间件全链路数据100%不丢失

第一个会导致数据丢失的地方，就是消费者获取到消息之后，没有来得及处理完毕，自己直接宕机了。这就要调整消费者的ack机制，通过在消费者服务中调整为手动ack机制，来确保消息一定是已经成功处理完了，才会发送ack通知给MQ集群。

第二个是集群接收到消息后整个集群直接宕机了。这个就要确认只要订单服务发送出去的消息确认成功了，此时MQ集群就一定已经将消息持久化到磁盘了。

其次，增加一个额外的系统，发送端成功发送消息给MQ后，同时发送这条消息的id到这个额外系统，服务端处理完这条消息后，将这条消息的状态更新为已处理。

建立死信MQ系统，确保原有MQ系统正常运行。

## 说说系统架构引入消息中间件有什么缺点？

系统可用性降低（MQ挂掉之后的高可用保障方案）

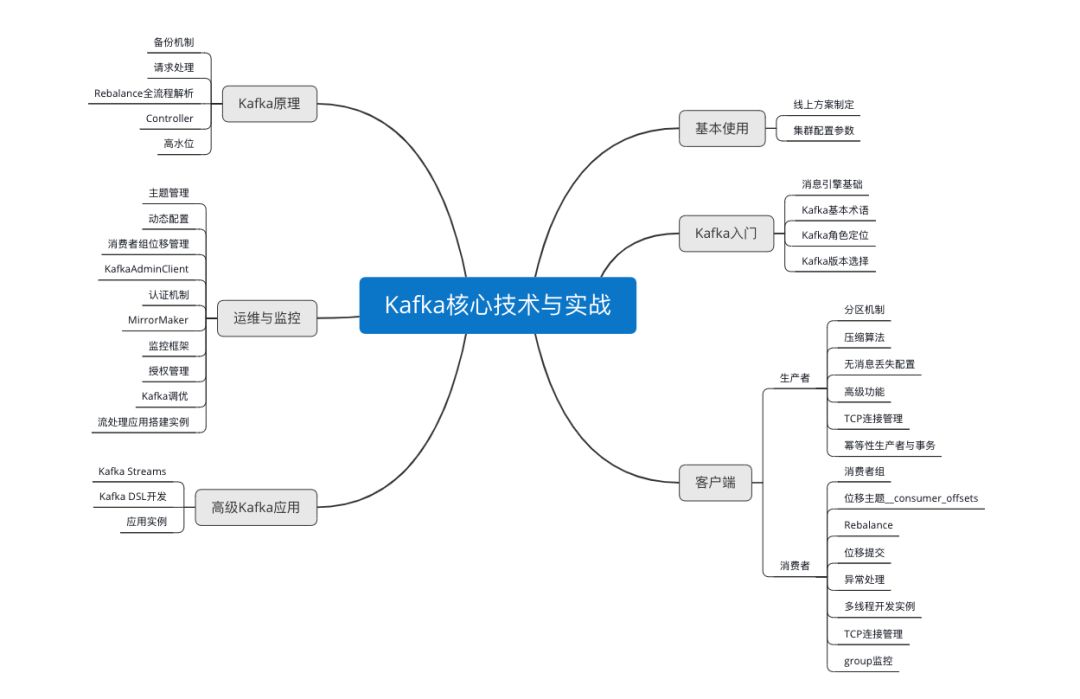
系统稳定性降低（MQ系统故障导致丢数据，多数据等问题）

MQ中间件的这些问题的解决方案，包括：消息高可靠传递（0丢失）；消息幂等性传递（绝对不重复）；百万消息积压的线上故障处理；

分布式一致性问题（C现在处理自己本地数据库成功了，然后发送了一个消息给MQ，系统D也确实是消费到了，但是结果不幸的是，系统D操作自己本地数据库失败了。）

# Kafka

Kafka 核心技术：我们仅需要学习一套框架，就能在实际业务系统中实现消息队列应用、应用程序集成、分布式存储构建，甚至是流处理应用的开发与部署



极客时间新上线的《Kafka 核心技术与实战》

crc校验：消息完整性校验

ack：一个确认反馈机制

ISR：leader中记录的与其保持同步的Replica列表。

## Kafka消费组

**什么是消费者组**

consumer group是kafka提供的可扩展且具有容错性的消费者机制,组内的所有消费者协调在一起来消费订阅主题(subscribed topics)的所有分区(partition)。当然，每个分区只能由同一个消费组内的一个consumer来消费。

消费者在消费的过程中需要记录自己消费了多少数据，即消费位置信息。即位移(offset)。

Kafka默认是定期帮你自动提交位移的(enable.auto.commit = true)，你当然可以选择手动提交位移实现自己控制。另外kafka会定期把group消费情况保存起来，做成一个offset map，

老版本的位移是提交到zookeeper中的，新版本的kafka提供了另一种解决方案：增加\_\_consumeroffsets topic，将offset信息写入这个topic，摆脱对zookeeper的依赖(指保存offset这件事情)。\_\_consumer\_offsets中的消息保存了每个consumer group某一时刻提交的offset信息。

## Kafka存储策略

1. Kafka以topic来进行消息管理，每个topic包含多个partitions，每个partition对应一个逻辑log，由多个segment组成。

2. 每个segment中存储多条消息，消息id由其逻辑位置决定，该id可直接定位到消息存储位置，避免id到位置的额外映射。

3．每个part(patition)在内存中对应一个index，记录每个segment中的第一条消息偏移offset。

4. 发布者发到某个topic的消息会被均匀的分布到多个part上，broker收到发布消息往对应part的最后一个segment上添加消息。当某个segment上的消息数达到配置或发布时间的阀值，会被flush到磁盘。

5．无论消息是否被消费，只要在配置的时间段内，消息都将被留。

## Kafka数据持久化

kafka使用文件存储消息(append only log),这就直接决定kafka在性能上严重依赖文件系统的本身特性.且无论任何OS下,对文件系统本身的优化是非常艰难的.文件缓存/直接内存映射等是常用的手段.因为kafka是对日志文件进行append操作,因此磁盘检索的开支是较小的;同时为了减少磁盘写入的次数,broker会将消息暂时buffer起来,当消息的个数(或尺寸)达到一定阀值时,再flush到磁盘,这样减少了磁盘IO调用的次数.对于kafka而言,较高性能的磁盘,将会带来更加直接的性能提升.

kafka是由follower周期性或者尝试去pull(拉)过来(其实这个过程与consumer消费过程非常相似)

**kafka不是完全同步，也不是完全异步，是一种ISR机制**

1. leader会维护一个与其基本保持同步的Replica列表，该列表称为ISR(in-sync Replica)，每个Partition都会有一个ISR，而且是由leader动态维护   
2. 如果一个flower比一个leader落后太多，或者超过一定时间未发起数据复制请求，则leader将其重ISR中移除   
3. 当ISR中所有Replica都向Leader发送ACK时，leader才commit

## Kfaka如何应对大量的长连接

Kafka采用Reactor多路复用模型来应对大量客户端连接（kafka会与生产者消费者建立大量的长连接，及其耗内存）

简单来说，就是搞一个acceptor线程，基于底层操作系统的支持，实现连接请求监听。

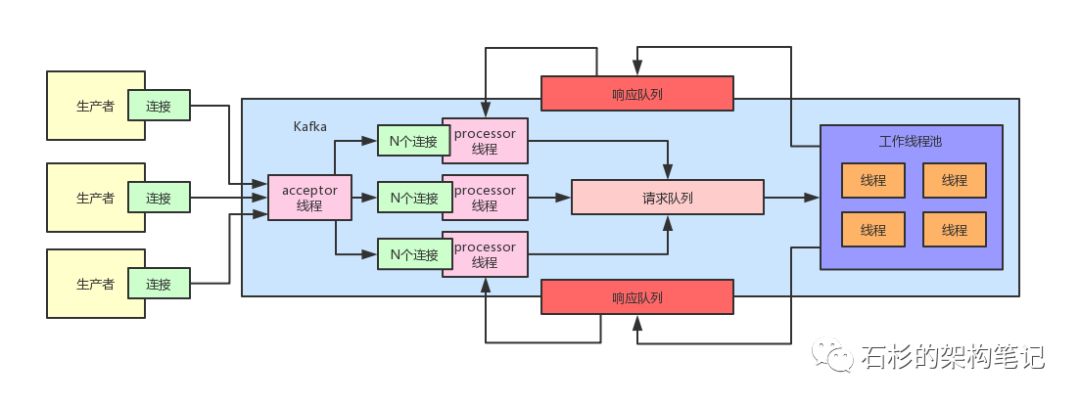
如果有某个设备发送了建立连接的请求过来，那么那个线程就把这个建立好的连接交给processor线程。

每个processor线程会被分配N多个连接，一个线程就可以负责维持N多个连接，他同样会基于底层操作系统的支持监听N多连接的请求。

如果某个连接发送了请求过来，那么这个processor线程就会把请求放到一个请求队列里去。

接着后台有一个线程池，这个线程池里有工作线程，会从请求队列里获取请求，处理请求，接着将请求对应的响应放到每个processor线程对应的一个响应队列里去。

最后，processor线程会把自己的响应队列里的响应发送回给客户端。



## Kafka消息丢失怎么处理？

消息丢失的情况

1.在producer发送了kafka未接到的情况：

设置ack=1的时候，表示producer发送出去message，同步的把message存到对应topic的partition的leader上，然后producer就返回成功，partition leader异步的把message同步到其他partition replica上。

2.在kafka发送了consumer未接收或未保存成功的情况：

enable.auto.commit=false  关闭自动提交位移

3.在kafka中丢失：

这种情况丢数据的方式也有很多种，比如：kafka controller 切换的时候，会导致partition leader的切换（老的 kafka controller上面的partition leader会选举到其他的kafka broker上）,但是这样就会导致丢数据。或者kafka上消息积压太多，导致很多消息过期。

使用kafka(0.8.2.1)高级API消费消息时，有时会因各种原因，导致消息堆积。如果请求offset对应消息已过期，则会抛出下面异常：

Current offset 789380 for partition [test,3] out of range; reset offset to 799380"

抛出该异常的同时，会把该topic和分区下次请求的offset重置为一个新的值，此时就发生了消息丢失。

那么，如果减少数据丢失呢？

注意异常后重置offset的值。先看处理offset超出有效范围的代码。

通过代码可以清楚看到，这个重置的offet值和auto.offset.reset配置有关。auto.offset.reset值有earliest和latest。当消费客户端启动时

**earliest** 当分区下有已提交的offset时，从提交的offset开始消费；无提交的offset时，从头开始消费。   
**latest** 当分区下有已提交的offset时，从提交的offset开始消费；无提交的offset时，消费新产生的该分区下的数据。

当消费客户端在消费过程中，如果有消息过期，earliest会将offset重置为kafka里最早消息，latest则会将offset重置为最新消息。因此，把auto.offset.reset设置为earliest， 可以在消费堆积的情况下，减少数据丢失。注意，auto.offset.reset默认是latest。

## Kafka的Leader选举机制？

Kakfa Broker集群受Zookeeper管理。所有的Kafka Broker节点一起去Zookeeper上注册一个临时节点，因为只有一个Kafka Broker会注册成功，其他的都会失败，所以这个成功在Zookeeper上注册临时节点的这个Kafka Broker会成为Kafka Broker Controller，其他的Kafka broker叫Kafka Broker follower。（这个过程叫Controller在ZooKeeper注册Watch）。这个Controller会监听其他的Kafka Broker的所有信息，如果这个kafka broker controller宕机了，在zookeeper上面的那个临时节点就会消失，此时所有的kafka broker又会一起去Zookeeper上注册一个临时节点，因为只有一个Kafka Broker会注册成功，其他的都会失败，所以这个成功在Zookeeper上注册临时节点的这个Kafka Broker会成为Kafka Broker Controller，其他的Kafka broker叫Kafka Broker follower。

## kafka如何保证消息的顺序

只有一个分区里的消息可以保证顺序，但是同一分区也可能乱序： 而且必须等上一条消息发送完再发送下一条消息。

假设a,b两条消息，a先发送后由于发送失败重试，这时顺序就会在b的消息后面，可以设置max.in.flight.requests.per.connection=1来避免 max.in.flight.requests.per.connection：

限制客户端在单个连接上能够发送的未响应请求的个数。设置此值是1表示kafka broker在响应请求之前client不能再向同一个broker发送请求，但吞吐量会下降

新启动的consumer默认从partition队列最头端最新的地方开始阻塞的读message

 Kafka producer 发送message不用维护message的offsite信息，因为这个时候，offsite就相当于一个自增id，producer就尽管发送message就好了。

Kafka的producer一般都是大批量的batch发送message

解释一下，在数据制作过程中，你如何能从Kafka得到准确的信息?

在数据中，为了精确地获得Kafka的消息，你必须遵循两件事: 在数据消耗期间避免重复，在数据生产过程中避免重复。

这里有两种方法，可以在数据生成时准确地获得一个语义:

每个分区使用一个单独的写入器，每当你发现一个网络错误，检查该分区中的最后一条消息，以查看您的最后一次写入是否成功

在消息中包含一个主键(UUID或其他)，并在用户中进行反复制

在数据消耗期间避免重复：

offset偏移量

## 为什么要使用kafka，有那种场景下需要使用jms

由于来不及同步处理，请求往往会发生堵塞，比如说，大量的insert，update之类的请求同时到达Mysql，直接导致无数的行锁表锁，甚至最后请求会堆积过多，从而触发too many connections错误。通过使用消息队列，我们可以异步处理请求，从而缓解系统的压力。

kafka是一个发布订阅消息系统，由topic区分消息种类，每个topic中可以有多个partition，每个kafka集群有一个多个broker服务器组成，producer可以发布消息到kafka中，consumer可以消费kafka中的数据。kafka就是生产者和消费者中间的一个暂存区，可以保存一段时间的数据保证使用。