

Kafka

• 核心知识：

• Kafka 文件存储 ==>



• 1. Kafka文件存储机制

• 名词解释 ==>

• Broker

- 消息中间件处理结点，一个Kafka节点就是一个broker，多个broker可以组成一个Kafka集群

• Topic

- 一类消息，例如page view日志、click日志等都可以以topic的形式存在，Kafka集群能够同时负责多个topic的分发

• Partition

- topic物理上的分组，一个topic可以分为多个partition，每个partition是一个有序的队列

• Segment

- partition物理上由多个segment组成

• offset

每个partition都由一系列有序的、不可变的消息组成，这些消息被连续的追加到partition中。partition中的每个消息都有一个连续的序列号叫做offset,用于partition唯一标识一条消息

• 1.1 Topic中 Partition 存储分布

假设实验环境中Kafka集群只有一个broker，xxx/message-folder为数据文件存储根目录，在Kafka broker中server.properties文件配置(参数log.dirs=xxx/message-folder)，例如创建2个topic名称分别为report_push、launch_info，partitions数量都为partitions=4

存储路径和目录规则为：

```
xxx/message-folder
|--report_push-0
|--report_push-1
|--report_push-2
|--report_push-3
|--launch_info-0
|--launch_info-1
|--launch_info-2
|--launch_info-3
```

在Kafka文件存储中，同一个topic下有多个不同partition，每个partition为一个目录，partiton命名规则为topic名称+有序序号，第一个partiton序号从0开始，序号最大值为partitions数量减1。

如果是多broker分布情况，请参考

• 1.2 Partiton 中文件存储方式

• 1.3 Partiton 中 Segment 文件存储结构

• 1.4 在 Partition 中如何通过 Offset 查找 Message

• 2. Kafka 消息发送的机制

每当用户往某个Topic发送数据时，数据会被hash到不同的partition,这些partition位于不同的集群节点上，所以每个消息都会被记录一个offset消息号，随着消息的增加逐渐增加，这个offset也会递增，同时，每个消息会有一个编号，就是offset号。消费者通过这个offset号去查询读取这个消息。

发送消息流程

1. 首先获取topic的所有Patition

2. 如果客户端不指定Patition，也没有指定Key的话，使用自增长的数字取余数的方式实现指定的Partition。这样Kafka将平均的向Partition中生产数据。

3. 如果想要控制发送的partition，则有两种方式，一种是指定partition，另一种就是根据Key自己写算法。继承Partitioner接口，实现其partition方法。

• 3. Kafka 消息消费机制

kafka 消费者有消费者族群的概念，当生产者将数据发布到topic时，消费者通过 pull 的方式，定期从服务器拉取数据,当然在pull数据的时候，服务器会告诉consumer可消费的消息offset。

创建一个Topic (名为topic1)，再创建一个属于group1的 Consumer实例，并创建三个属于group2 的Consumer实例，然后通过 Producer向topic1发送Key分别为1，2，3的消息。结果发现属于group1的Consumer收到了所有的这三条消息，同时 group2中的3个Consumer分别收到了Key为1，2，3的消息，如下图所示。

(在 Storm 中使用 client_id 的意义)

结论：不同 Consumer Group下的消费者可以消费partition中相同的消息，相同的Consumer Group下的消费者只能消费partition中不同的数据。

服务器会记录每个consumer的在每个topic的每个partition下的消费的offset,然后每次去消费去拉取数据时，都会从上次记录的位置开始拉取数据。比如0.8版本的用zookeeper来记录

/{consumer}/{group_name}/{id}/{consumer_id} //记录id

/{consumer}/{group_name}/{offset}/{topic_name}/{partitions_id} //记录偏移量

/{consumer}/{group_name}/{owner}/{topic_name}/{partitions_id} //记录分区属于哪个消费者

当consumer和partition增加或者删除时,需要重新执行一遍 Consumer Rebalance算法

Consumer Rebalance的算法如下 ==>

将目标Topic下的所有Partirtion排序，存于PT

对某Consumer Group下所有Consumer排序，存于CG，第i个Consumer记为Ci

N=size(PT)/size(CG)，向上取整

解除Ci对原来分配的Partition的消费权 (i从0开始)

将第i*N到 (i+1) *N-1个Partition分配给Ci

• 4. Kafka 消息存储机制

kafka的消息是存储在磁盘的，所以数据不易丢失，如上了解，partition是存放消息的基本单位，那么它是如何存储在文件当中的呢，如上：topic-partition-id，每个partition都会保存成一个文件，这个文件又包含两部分。 .index索引文件、.log消息内容文件。

index文件结构很简单，每一行都是一个key,value对

key 是消息的序号offset，value 是消息的物理位置偏移量。index索引文件 (offset消息编号-消息在对应文件中的偏移量)

比如：要查找绝对offset为7的 Message：

1. 首先是用二分查找确定它是在哪个LogSegment中，自然是在第一个Segment中。

2. 打开这个Segment的index文件，也是用二分查找找到offset小于或者等于指定offset的索引条目中最大的那个offset。自然offset为6的那个索引是我们要找的，通过索引文件我们知道offset为6的Message在数据文件中的位置为9807。

3. 打开数据文件，从位置为9807的那个地方开始顺序扫描直到找到offset为7的那条Message。

这套机制是建立在offset是有序的。索引文件被映射到内存中，所以查找的速度还是很快的。

这是一种 稀疏索引文件机制，并没有把每个消息编号和文件偏移量记录下来，而是稀疏记录一部分，这样可以方式索引文件占据过多空间。每次查找消息时，需要将整块消息读入内存，然后获取对应的消息。

比如消息offset编号在36,37,38的消息,都会通过38找到对应的offset

● 5. 日志更新和清理

Kafka中如果消息有key，相同key的消息在不同时刻有不同的值，则只允许存在最新的一条消息，这就好比传统数据库的 update 操作，查询结果一定是最近update的那一条，而不应该查询出多条或者查询出旧的记录，当然对于HBase/Cassandra这种支持多版本的数据库而言，update操作可能导致添加新的列，查询时是合并的结果而不一定是最新的记录。图3-27中示例了多条消息，一旦key已经存在，相同key的旧的消息会被删除，新的被保留。如下图，就是对日志更新然后压缩。

https://static.oschina.net/uploads/space/2017/0227/163837_Fls4_1403215.png

清理后Log Head部分每条消息的offset都是逐渐递增的，而Tail部分消息的offset是断断续续的。LogToClean 表示需要被清理的日志

https://static.oschina.net/uploads/space/2017/0227/185108_GXPP_1403215.png

生产者客户端如果发送的消息key的value是空的，表示要删除这条消息，发生在删除标记之前的记录都需要删除掉，而发生在删除标记(Cleaner Point)之后的记录则不会被删除。

● 6. 消息检索过程示例 ==>

● Demo - 读取offset=368的消息：

● 1) 找到第368条消息在哪个segment

从partition目录中取得所有segment文件的名称，就相当于得到了各个序号区间

例如有3个segment

000000000000000000000000.index

000000000000000000000000.log

000000000000000000000300.index

000000000000000000000300.log

000000000000000000000600.index

000000000000000000000600.log

根据二分查找，可以快速定位，第368条消息是在0000000000000000000300.log文件中

● 2) 从index文件中找到其物理偏移量

读取 0000000000000000000300.index 。以368为key，得到value，如299，就是消息的物理位置偏移量

● 3) 到log文件中读取消息内容

读取 0000000000000000000300.log 从偏移量299开始读取消息内容。完成了消息的检索过程

● Kafka Streaming

● Kafka 的设计解析

● 1、Kafka背景及架构介绍



● 2、Kafka High Availability （上）



● 3、Kafka High Availability （下）



● 4、Kafka Consumer设计解析



● 5、Kafka性能测试方法及Benchmark报告



● 6、Kafka高性能架构之道



● 宏观架构层面

● 1、利用Partition实现并行处理

● Partition提供并行处理的能力

● Partition是最小并发粒度

● 2、ISR实现可用性与数据一致性的动态平衡

● CAP理论

CAP理论是指，分布式系统中，一致性、可用性和分区容忍性最多只能同时满足两个。

一般而言，都要求保证分区容忍性。所以在CAP理论下，更多的是需要在可用性和一致性之间做权衡。

● 一致性

● 通过某个节点的写操作结果对后面通过其它节点的读操作可见

● 如果更新数据后，并发访问情况下后续读操作可立即感知该更新，称为强一致性

● 如果允许之后部分或者全部感知不到该更新，称为弱一致性

● 若在之后的一段时间（通常该时间不固定）后，一定可以感知到该更新，称为最终一致性

● 可用性

- 任何一个没有发生故障的节点必须在有限的时间内返回合理的结果
 - 分区容忍性
 - 部分节点宕机或者无法与其它节点通信时，各分区间还可保持分布式系统的功能
- 常用数据复制及一致性方案
 - Master-Slave
 - WNR
 - Paxos及其变种
 - 基于ISR的数据复制方案
- 使用ISR方案的原因
 - 由于Leader可移除不能及时与之同步的Follower，故与同步复制相比可避免最慢的Follower拖慢整体速度，也即ISR提高了系统可用性。
 - ISR中的所有Follower都包含了所有Commit过的消息，而只有Commit过的消息才会被Consumer消费，故从Consumer的角度而言，ISR中的所有Replica都始终处于同步状态，从而与异步复制方案相比提高了数据一致性。
 - ISR可动态调整，极限情况下，可以只包含Leader，极大提高了可容忍的宕机的Follower的数量。与Majority Quorum方案相比，容忍相同个数的节点失败，所要求的总节点数少了近一半。
- ISR相关配置说明
 1. Broker的min.insync.replicas参数指定了Broker所要求的ISR最小长度，默认值为1。也即极限情况下ISR可以只包含Leader。但此时如果Leader宕机，则该Partition不可用，可用性得不到保证。
 2. 只有被ISR中所有Replica同步的消息才被Commit，但Producer发布数据时，Leader并不需要ISR中的所有Replica同步该数据才确认收到数据。Producer可以通过acks参数指定最少需要多少个Replica确认收到该消息才视为该消息发送成功。acks的默认值是1，即Leader收到该消息后立即告诉Producer收到该消息，此时如果在ISR中的消息复制完该消息前Leader宕机，则该消息会丢失。而如果将该值设置为0，则Producer发送完数据后，立即认为该数据发送成功，不作任何等待，而实际上该数据可能发送失败，并且Producer的Retry机制将不生效。更推荐的做法是，将acks设置为all或者-1，此时只有ISR中的所有Replica都收到该数据（也即该消息被Commit），Leader才会告诉Producer该消息发送成功，从而保证不会有未知的数据丢失。
- 具体实现层面
 - 高效使用磁盘
 - 顺序写磁盘
 - 充分利用Page Cache

Broker收到数据后，写磁盘时只是将数据写入Page Cache，并不保证数据一定完全写入磁盘。从这一点看，可能会造成机器宕机时，Page Cache内的数据未写入磁盘而造成数据丢失。但是这种丢失只发生在机器断电等造成操作系统不工作的场景，而这种场景完全可以由Kafka层面的Replication机制去解决。如果为了保证这种情况下数据不丢失而强制将Page Cache中的数据Flush到磁盘，反而会降低性能。也正因如此，Kafka虽然提供了flush.messages和flush.ms两个参数将Page Cache中的数据强制Flush到磁盘，但是Kafka并不建议使用。如果数据消费速度与生产速度相当，甚至不需要通过物理磁盘交换数据，而是直接通过Page Cache交换数据。同时，Follower从Leader Fetch数据时，也可通过Page Cache完成。

 - 使用Page Cache的好处如下
 - - I/O Scheduler会将连续的小块写组装成大块的物理写从而提高性能
 - - I/O Scheduler会尝试将一些写操作重新按顺序排好，从而减少磁盘头的移动时间
 - - 充分利用所有空闲内存（非JVM内存）。如果使用应用层Cache（即JVM堆内存），会增加GC负担
 - - 读操作可直接在Page Cache内进行。如果消费和生产速度相当，甚至不需要通过物理磁盘（直接通过Page Cache）交换数据
 - - 如果进程重启，JVM内的Cache会失效，但Page Cache仍然可用
 - 某Partition的Leader节点的网络/磁盘读写信息

从上图可以看到，该Broker每秒通过网络从Producer接收约35MB数据，虽然有Follower从该Broker Fetch数据，但是该Broker基本无读磁盘。这是因为该Broker直接从Page Cache中将数据取出返回给了Follower。

```

---total-cpu-usage--- -dsk/total- -net/total-
usr sys idl wai hiq siq | read writ | rcv send
0 0 0 100 0 0 | 80B 19k | 0 0
4 4 91 0 0 0 | 0 0 | 41M 293k
4 4 92 0 0 0 | 0 0 | 40M 296k
6 4 90 0 0 0 | 0 0 | 40M 309k
5 5 90 0 0 0 | 52k 42M 316k
4 4 92 0 0 0 | 0 33M 271k
2 3 92 4 0 0 | 113M 13M 121k
3 5 78 14 0 0 | 718M 21M 183k
1 2 97 0 0 0 | 104k 8136k 97k
3 3 93 1 0 0 | 25M 25M 184k
4 4 92 0 0 0 | 0 39M 285k
3 4 93 0 0 0 | 0 27M 196k

```
 - 支持多Disk Drive
 - Broker的log.dirs配置项，允许配置多个文件夹

支持多Disk Drive
Broker的log.dirs配置项，允许配置多个文件夹。如果机器上有多个Disk Drive，可将不同的Disk挂载到不同的目录，然后将这些目录都配置到log.dirs里。Kafka会尽可能将不同的Partition分配到不同的目录，也即不同的Disk上，从而充分利用了多Disk的优势。
 - 零拷贝

Kafka中存在大量的网络数据持久化到磁盘（Producer到Broker）和磁盘文件通过网络发送（Broker到Consumer）的过程。这一过程的性能直接影响Kafka的整体吞吐量。

 - 传统模式下的四次拷贝与四次上下文切换
 - sendfile和transferTo实现零拷贝
 - 减少网络开销
 - 批处理

批处理是一种常用的用于提高I/O性能的方式。对Kafka而言，批处理既减少了网络传输的Overhead，又提高了写磁盘的效率。Kafka 0.8.1及以前的Producer区分同步Producer和异步Producer。同步Producer的send方法主要分两种形式。一种是接受一个KeyedMessage作为参数，一次发送一条消息。另一种是接受一批KeyedMessage作为参数，一次性发送多条消息。而对于异步发送而言，无论是使用哪个send方法，实现上都不会立即将消息发送给Broker，而是先存到内部的队列中，直到消息条数达到阈值或者达到指定的Timeout才真正的将消息发送出去，从而实现了消息的批量发送。Kafka 0.8.2开始支持新的Producer API，将同步Producer和异步Producer结合。虽然从send接口来看，一次只能发送一个ProducerRecord，而不能像之前版本的send方法一样接受消息列表，但是send方法并非立即将消息发送出去，而是通过batch.size和linger.ms控制实际发送频率，从而实现批量发送。由于每次网络传输，除了传输消息本身以外，还要传输非常多的网络协议本身的一些内容（称为Overhead），所以将多条消息合并到一起传输，可有效减少网络传输的Overhead，进而提高了传输效率。从 中可以看到，虽然Broker持续从网络接收数据，但是写磁盘并非每秒都在发生，而是间隔一段时间写一次磁盘，并且每次写磁盘的数据量都非常大（最高达到718MB/S）。

- 数据压缩降低网络负载

Kafka从0.7开始，即支持将数据压缩后再传输给Broker。除了可以将每条消息单独压缩然后传输外，Kafka还支持在批量发送时，将整个Batch的消息一起压缩后传输。数据压缩的一个基本原理是，重复数据越多压缩效果越好。因此将整个Batch的数据一起压缩能更大幅度减小数据量，从而更大程度提高网络传输效率。Broker接收消息后，并不直接解压缩，而是直接将消息以压缩后的形式持久化到磁盘。Consumer Fetch到数据后再解压缩。因此Kafka的压缩不仅减少了Producer到Broker的网络传输负载，同时也降低了Broker磁盘操作的负载，也降低了Consumer与Broker间的网络传输量，从而极大得提高了传输效率，提高了吞吐量。

- 高效的序列化方式

Kafka消息的Key和Payload（或者说Value）的类型可自定义，只需同时提供相应的序列化器和反序列化器即可。因此用户可以通过使用快速且紧凑的序列化-反序列化方式（如Avro, Protocol Buffer）来减少实际网络传输和磁盘存储的数据规模，从而提高吞吐率。这里要注意，如果使用的序列化方法太慢，即使压缩比非常高，最终的效率也不一定高。

- Kafka 技术内幕（未上市-初稿）



- High Level Overview



- C1-C4 知识点



- 第一章 Kafka入门
 - 1.1 介绍
 - 1.1.1 流式数据平台
 - 1.1.2 主要概念
 - 1.1.3 Kafka的设计实现
 - 1.2 快速开始
 - 1.2.1 单机模式
 - 1.2.2 集群模式
 - 1.2.3 消费组示例
 - 1.3 环境准备
 - 1.3.1 编译运行
 - 1.3.2 本书导读
 - 第二章 生产者
 - 2.1 新生产者客户端
 - 2.1.1 同步和异步发送消息
 - 2.1.2 客户端消息发送线程
 - 2.1.3 客户端网络连接对象
 - 2.1.4 选择器处理网络请求
 - 2.2 旧生产者客户端
 - 2.2.1 事件处理器处理客户端发送的消息
 - 2.2.2 对消息集按照节点和分区进行整理
 - 2.2.3 生产者使用阻塞通道发送请求
 - 2.3 服务端网络连接
 - 2.3.1 服务端使用接收器接受客户端的连接
 - 2.3.2 处理器使用选择器的轮询处理网络请求
 - 2.3.3 请求通道的请求队列和响应队列
 - 2.3.4 Kafka请求处理线程
 - 2.3.5 服务端的请求处理入口
 - 2.4 小结
 - 第三章：消费者（高级和低级API）
 - 3.1 消费者启动和初始化
 - 3.1.1 创建并初始化消费者连接器
 - 3.1.2 消费者客户端的线程模型
 - 3.1.3 重新初始化消费者
 - 3.2 消费者再平衡操作
 - 3.2.1 分区的所有权
 - 3.2.2 为消费者分配分区
 - 3.2.3 创建分区信息对象
 - 3.2.4 关闭和更新拉取线程管理器
 - 3.2.5 分区信息对象的偏移量
 - 3.3 消费者拉取数据
 - 3.3.1 拉取线程管理器
 - 3.3.2 抽象拉取线程
 - 3.3.3 消费者拉取线程
 - 3.4 消费者消费消息
 - 3.4.1 Kafka消息流
 - 3.4.2 消费者迭代消费消息
 - 3.5 消费者提交分区偏移量
 - 3.5.1 提交偏移量到ZooKeeper
 - 3.5.2 提交偏移量到内部主题
 - 3.5.3 连接偏移量管理器
 - 3.5.4 服务端处理提交偏移量的请求
 - 3.5.5 缓存分区的偏移量
 - 3.6 消费者低级API示例
 - 3.6.1 消息消费主流程
 - 3.6.2 找出分区的Leader
 - 3.6.3 获取分区的读取偏移量
 - 3.6.4 发送拉取请求并消费消息
 - 3.7 小结
 - 第四章 新消费者
 - 4.1 新消费者客户端
 - 4.1.1 消费者的订阅状态
 - 4.1.2 消费者轮询的准备工作
 - 4.1.3 消费者轮询的流程
 - 4.1.4 消费者拉取消息
 - 4.1.5 消费者获取记录
 - 4.1.6 消费消息
 - 4.2 消费者的网络客户端轮询
 - 4.2.1 异步请求
 - 4.2.2 异步请求高级模式
 - 4.2.3 网络客户端轮询
 - 4.3 心跳任务
 - 4.3.1 发送心跳请求
 - 4.3.2 心跳状态
 - 4.3.3 运行心跳任务
 - 4.3.4 处理心跳结果的示例
 - 4.3.5 心跳和协调者的关系
 - 4.4 消费者提交偏移量
 - 4.4.1 自动提交任务
 - 4.4.2 将拉取偏移量作为提交偏移量
 - 4.4.2 同步提交偏移量
 - 4.4.3 消费者的消息处理语义
 - 4.5 小结

- C5-C9 知识点



- 第五章 协调者
 - 5.1 消费者加入消费组
 - 5.1.1 元数据与分区分配器
 - 5.1.2 消费者的加入组和同步组
 - 5.1.3 Leader消费者执行分配任务
 - 5.1.4 加入组的准备、完成和监听器
 - 5.2 协调者处理请求
 - 5.2.1 服务端定义发送响应结果的回调方法
 - 5.2.2 消费者和消费组元数据
 - 5.2.3 协调者处理请求前的条件检查
 - 5.2.4 协调者调用回调方法发送响应给客户端
 - 5.3 延迟的加入组操作
 - 5.4 消费组状态机
 - 5.4.1 再平衡操作与监听器
 - 5.4.2 消费组的状态转换
 - 5.4.3 协调者处理“加入组请求”
 - 5.4.4 协调者处理“同步组请求”
 - 5.4.5 协调者处理“离开组请求”
 - 5.4.6 再平衡超时与会话超时
 - 5.4.7 延迟的心跳
 - 5.5 小结
- 第六章 存储层
 - 6.1 日志的读写
 - 6.1.1 分区、副本、日志、日志分段
 - 6.1.2 写入日志
 - 6.1.3 日志分段
 - 6.1.4 读取日志
 - 6.1.5 日志管理
 - 6.1.5 日志压缩
 - 6.2 服务端处理读写请求
 - 6.2.1 副本管理器
 - 6.2.2 分区与副本
 - 6.3 延迟操作
 - 6.3.1 延迟操作接口
 - 6.3.2 延迟操作与延迟缓存
 - 6.3.3 延迟缓存
 - 6.4 小结
- 第七章 控制器
 - 7.1 Kafka控制器
 - 7.1.1 控制器选举
 - 7.1.2 控制器上下文
 - 7.1.3 ZK监听器
 - 7.1.4 分区状态机和副本状态机
 - 7.1.5 删除主题
 - 7.1.6 重新分配分区
 - 7.1.7 控制器的网络通道管理器
 - 7.2 服务端处理 LeaderAndISR 请求
 - 7.2.1 创建分区
 - 7.2.2 创建主副本、从副本
 - 7.2.3 消费组元数据迁移
 - 7.3 元数据缓存
 - 7.3.1 服务端的元数据缓存
 - 7.3.2 客户端更新元数据
 - 7.4 Kafka服务关闭
 - 7.5 小结
- 第八章 基于Kafka构建数据流管道
 - 8.1 Kafka集群同步工具 (MirrorMaker)
 - 8.1.1 单机模拟数据同步
 - 8.1.2 数据同步的流程
 - 8.2. Uber集群同步工具 (uReplicator)
 - 8.2.1 Apache Helix介绍
 - 8.2.2 Helix控制器
 - 8.2.3 Helix工作节点
 - 8.3 Kafka连接器
 - 8.3.1 连接器的使用示例
 - 8.3.2 开发一个简单的连接器
 - 8.3.3 连接器的架构模型
 - 8.3.4 Herder的实现
 - 8.3.5 Worker 的实现
 - 8.3.6 配置存储与状态存储
 - 8.3.7 连接器与任务的实现
 - 8.4 小结
- 第九章 Kafka流处理
 - 9.1 低级 Processor API
 - 9.1.1 流处理应用程序示例
 - 9.1.2 流处理的拓扑
 - 9.1.3 流处理的线程模型
 - 9.1.4 状态存储
 - 9.2 高级流式 DSL
 - 9.2.1 DSL 应用程序示例
 - 9.2.2 KStream 和 KTable
 - 9.2.3 连接操作
 - 9.2.4 窗口操作
 - 9.3 小结

• C2_生产者

• Kafka 应用场景 《Apache Kafka Cookbook》

• C1- Kafka 的启动

- 1、搭建多个 Kafka Brokers
- 2、创建 Topics

- `> bin/kafka-topics.sh --create --ZooKeeper localhost:2181 --replication-factor 1 --partitions 1 --topic kafkatest`

- 检查 Topic 是否成功创建:

- `> bin/kafka-topics.sh --list --ZooKeeper localhost:2181`
- kafkatest

- 获取指定 Topic 的详细信息:

- `> bin/kafka-topics.sh --describe --ZooKeeper localhost:2181 --topic kafkatest`
- Topic:kafkatest PartitionCount:1 ReplicationFactor:1 Configs:
- Topic: kafkatest Partition: 0 Leader: 0 Replicas: 0 Isr: 0

• 3、从终端发送信息

- 执行命令:

- `> bin/kafka-console-producer.sh --broker-list localhost:9092 --topic kafkatest`

- 参数详解:

- `--broker-list`: This specifies the ZooKeeper servers. It is followed by a list of the ZooKeeper server's hostname and port number that can be separated by a comma.
- `--topic`: This specifies the name of the topic. The name of the topic follows this parameter.
- `--sync`: This specifies that the messages should be sent synchronously—one at a time as they survive.
- `--compression-codec`: This specifies the compression codec that will be used to produce the messages. It can be none, gzip, snappy, or lz4. If it is not specified, it will by default be set to gzip.
- `--batch-size`: This specifies the number of messages to be sent in a single batch in case they are not being sent synchronously. This is followed by the batch's size value.
- `--message-send-max-retries`: Brokers can sometimes fail to receive messages for a number of reasons and being unavailable transiently is just one of them. This property specifies the number of retries before a producer gives up and drops the message. This is followed by the number of retries that you want to set.
- `--retry-backoff-ms`: Before each retry, the producer refreshes the metadata of relevant topics. Since leader election might take some time, it's good to specify some time before producer retries. This parameter does just that. This follows the time in ms.

- 4、在终端消费信息

- 执行命令：

- > bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic kafkatest --from-beginning

- 参数详解：

- fetch-size: This specifies the amount of data to be fetched in a single request. Its size in bytes follows this argument.
 - socket-buffer-size: This specifies the size of the TCP RECV size. The size in bytes follows the argument.
 - autocommit.interval.ms: This specifies the time interval in which the current offset is saved in ms. The time in ms follows the argument.
 - max-messages: This specifies the maximum number of messages to consume before exiting. If it is not set, the consumption is unlimited. The number of messages follows the argument.
 - skip-message-on-error: This specifies that, if there is an error while processing a message, the system should not stop. Instead, it should just skip the current messages.

- C2 - 配置 Brokers

- 1、配置基本信息

- broker.id=0
 - host.name=localhost
 - port=9092
 - log.dirs=/disk1/kafka-logs
 - advertised.host.name

advertised.host.name。官方文档里的备注信息表明，该字段的值是生产者和消费者使用的。如果没有设置，则会取host.name的值，默认情况下，该值为localhost。思考一下，如果生产者拿到localhost这个值，只往本地发消息，必然会报错（因为本地没有kafka服务器）另外，在配置生产者时，metadata.broker.list会设置成kafka服务器的IP和地址。但这个只是获取一些元信息，后续发送消息时会根据获取的元信息来发送，而获取元信息中，由于advertised.host.name被默认为localhost，所以本地当然会把消息发到本地，结果导致问题出现
<http://www.mamicode.com/info-detail-1008759.html>

- advertised.port

默认值：Null

分发这个端口给所有的producer，consumer和其他broker来建立连接。如果此端口跟server绑定的端口不同，则才有必要设置

- 2、配置线程 和 性能

- message.max.bytes = 1000000
 - broker能接收消息的最大字节数，这个值应该比消费端的fetch.message.max.bytes更小才对，否则broker就会因为消费端无法使用这个消息而挂起
 - num.network.threads = 3
 - broker处理消息的最大线程数，一般情况下不需要去修改
 - num.io.threads = 8
 - broker处理磁盘IO的线程数，数值应该大于你的硬盘数
 - background.threads = 10
 - 一些后台任务处理的线程数，例如过期消息文件的删除等，一般情况下不需要去做修改
 - queued.max.requests = 500
 - 等待IO线程处理的请求队列最大数，若是等待IO的请求超过这个数值，那么会停止接受外部消息，应该是一种自我保护机制。
 - socket.send.buffer.bytes = 102400
 - socket的发送缓冲区，socket的调优参数SO_SNDBUFF
 - socket.receive.buffer.bytes = 102400
 - socket的接受缓冲区，socket的调优参数SO_RCVBUFF
 - socket.request.max.bytes = 104857600(100*1024*1024)
 - socket请求的最大数值，防止serverOOM，message.max.bytes必然要小于socket.request.max.bytes，会被topic创建时的指定参数覆盖
 - num.partitions = 1
 - 每个topic的分区个数，若是在topic创建时候没有指定的话会被topic创建时的指定参数覆盖

- 3、日志配置（server.properties）

- log.segment.bytes = 1073741824 (1024*1024*1024)
 - topic的分区是以一堆segment文件存储的，这个控制每个segment的大小，会被topic创建时的指定参数覆盖
 - log.roll.hours = 168
 - 这个参数会在日志segment没有达到log.segment.bytes设置的大小，也会强制新建一个segment会被topic创建时的指定参数覆盖
 - log.cleanup.policy = delete
 - 日志清理策略选择有：delete和compact主要针对过期数据的处理，或是日志文件达到限制的额度，会被topic创建时的指定参数覆盖
 - log.retention.hours = 168

- `log.retention.minutes = 3days`
 - 数据存储的最大时间超过这个时间会根据`log.cleanup.policy`设置的策略处理数据，也就是消费端能够多久去消费数据
- `log.retention.bytes = -1`
 - `topic`每个分区的最大文件大小，一个`topic`的大小限制 = 分区数*`log.retention.bytes`。-1没有大小限制
`log.retention.bytes`和`log.retention.minutes`任意一个达到要求，都会执行删除，会被`topic`创建时的指定参数覆盖
- `log.retention.check.interval.ms = 30000`
 - 文件大小检查的周期时间，是否处罚 `log.cleanup.policy`中设置的策略
- `log.cleaner.enable = false`
 - 是否开启日志压缩
- `log.cleaner.threads = 1`
 - 日志压缩运行的线程数
- `log.cleaner.backoff.ms = 15000`
 - 检查是否处罚日志清理的间隔
- `log.index.size.max.bytes = 10485760`
 - 对于`segment`日志的索引文件大小限制，会被`topic`创建时的指定参数覆盖
- `log.index.interval.bytes = 4096`
 - 当执行一个`fetch`操作后，需要一定的空间来扫描最近的`offset`大小，设置越大，代表扫描速度越快，但是也更好内存，一般情况下不需要搭理这个参数
- `log.flush.interval.messages = Long.MaxValue`

log文件"sync"到磁盘之前累积的消息条数,因为磁盘IO操作是一个慢操作,但又是一个"数据可靠性"的必要手段,所以此参数的设置,需要在"数据可靠性"与"性能"之间做必要的权衡.如果此值过大,将会导致每次"fsync"的时间较长(IO阻塞),如果此值过小,将会导致"fsync"的次数较多,这也意味着整体的client请求有一定的延迟.物理server故障,将会导致没有fsync的消息丢失.
- `log.flush.interval.ms = Long.MaxValue`

仅仅通过interval来控制消息的磁盘写入时机,是不足的.此参数用于控制"fsync"的时间间隔,如果消息量始终没有达到阈值,但是离上一次磁盘同步的时间间隔达到阈值,也将触发.
- 4、配置副本策略 (`server.properties`)
 - `default.replication.factor = 1`
 - 是否允许自动创建`topic`，若是`false`，就需要通过命令创建`topic`
 - `replica.lag.time.max.ms = 10000`
 - `replicas`响应`partition leader`的最长等待时间，若是超过这个时间，就将`replicas`列入`ISR(in-sync replicas)`，并认为它是死的，不会再加入管理中
 - `replica.lag.max.messages = 4000`
 - 如果`follower`落后与`leader`太多,将会认为此`follower`[或者说`partition replicas`]已经失效
 - ##通常,在`follower`与`leader`通讯时,因为网络延迟或者链接断开,总会导致`replicas`中消息同步滞后
 - ##如果消息之后太多,leader将认为此`follower`网络延迟较大或者消息吞吐能力有限,将会把此`replicas`迁移
 - ##到其他`follower`中.
 - ##在`broker`数量较少,或者网络不足的环境中,建议提高此值.
 - `replica.fetch.max.bytes = 1048576 (1024 * 1024)`
 - `replicas`每次获取数据的最大大小
 - `replica.fetch.wait.max.ms = 500`
 - `replicas`同`leader`之间通信的最大等待时间，失败了会重试
 - `num.replica.fetchers = 1`
 - `leader`进行复制的线程数，增大这个数值会增加`follower`的IO
 - `replica.high.watermark.checkpoint.interval.ms = 5000`
 - 每个`replica`检查是否将最高水位进行固化的频率
 - `fetch.purgatory.purge.interval.requests = 1000`
 - `fetch` 请求清除时的清除间隔
 - `producer.purgatory.purge.interval.requests = 1000`
 - `producer`请求清除时的清除间隔
 - `replica.socket.timeout.ms = 30000 (30*1000)`
 - `leader` 备份数据时的`socket`网络请求的超时时间
 - `replica.socket.receive.buffer.bytes = 65536 (64*1024)`
 - 备份时向`leader`发送网络请求时的`socket receive buffer`
- 5、Zookeeper 的配置 (`server.properties`)
 - `server.properties = 127.0.0.1:2181,192.168.0.32:2181`
 - `zookeeper`集群的地址，可以是多个，多个之间用逗号分割
`hostname1:port1,hostname2:port2,hostname3:port3`

- `zookeeper.session.timeout.ms = 6000`
 - ZooKeeper的最大超时时间，就是心跳的间隔，若是没有反映，那么认为已经死了，不易过大
- `zookeeper.connection.timeout.ms = 6000`
 - ZooKeeper的连接超时时间
- `zookeeper.sync.time.ms = 2000`
 - ZooKeeper集群中leader和follower之间的同步实际那
- 6、其他杂项参数
 - `auto.create.topics.enable=true`
 - `controlled.shutdown.enable=true`
 - `controlled.shutdown.max.retries=3`
 - `controlled.shutdown.retry.backoff.ms=5000`
 - `auto.leader.rebalance.enable=true`
 - `leader.imbalance.per.broker.percentage=10`
 - `leader.imbalance.check.interval.seconds=300`
 - `offset.metadata.max.bytes=4096`
 - `max.connections.per.ip=Int.MaxValue`
 - `connections.max.idle.ms=600000`
 - `unclean.leader.election.enable=true`
 - `offsets.topic.num.partitions=50`
 - `offsets.topic.retention.minutes=1440`
 - `offsets.retention.check.interval.ms=600000`
 - `offsets.topic.replication.factor=3`
 - `offsets.topic.segment.bytes=104857600`
 - `offsets.load.buffer.size=5242880`
 - `offsets.commit.required.acks=-1`
 - `offsets.commit.timeout.ms=5000`
- C3 - 配置生产者 & 消费者 （待翻译）
 - 1、配置生产者的基本信息
 - `metadata.broker.list=192.168.0.2:9092,192.168.0.3:9092`
 - `request.required.acks=0`
 - `request.timeout.ms=10000`
 - 2、配置生产者的线程和性能
 - `serializer.class=kafka.serializer.DefaultEncoder`
 - `key.serializer.class=kafka.serializer.DefaultEncoder`
 - `partitioner.class=kafka.producer.DefaultPartitioner`
 - `compression.codec=none`
 - `compressed.topics=mytesttopic1`
 - `message.send.max.retries=3`
 - `retry.backoff.ms=100`
 - `topic.metadata.refresh.interval.ms=600000`
 - `producer.type=sync`
 - `queue.buffering.max.ms=5000`
 - `queue.buffering.max.messages=10000`
 - `queue.enqueue.timeout.ms=-1`
 - `batch.num.messages=200`
 - `send.buffer.bytes=102400`
 - `client.id=my_client`
 - 3、配置消费者的基本信息
 - `group.id=mygid`
 - `zookeeper.connect=192.168.0.2:2181`
 - `consumer.id=mycid`
 - 4、配置消费者的线程和性能
 - `socket.timeout.ms=30000`
 - `socket.receive.buffer.bytes=65536`

- `fetch.message.max.bytes=1048576`
 - `queued.max.message.chunks=2`
 - `fetch.min.bytes=1`
 - `fetch.wait.max.ms=100`
 - `consumer.timeout.ms=-1`
- 5、Consumer 的日志配置
 - `auto.commit.enable=true`
 - `auto.commit.interval.ms=60000`
 - `rebalance.max.retries=4`
 - `rebalance.backoff.ms=2000`
 - `refresh.leader.backoff.ms=200`
 - `auto.offset.reset=largest`
 - `partition.assignment.strategy=range`
- 6、Consumer 的 Zookeeper 配置
 - `zookeeper.session.timeout.ms=6000`
 - `zookeeper.connection.timeout.ms=6000`
 - `zookeeper.sync.time.ms=2000`
- 7、Consumer 的其他配置
 - `offsets.storage=zookeeper`
 - `offsets.channel.backoff.ms=6000`
 - `offsets.channel.socket.timeout.ms=6000`
 - `offsets.commit.max.retries=5`
 - `dual.commit.enabled=true`
 - `client.id=mycid`
- C4 - Kafka 的管理
 - 1、检查 Consumer 的 Offset
 - `$ bin/kafka-run-class.sh kafka.tools.ConsumerOffsetChecker --broker-info --zookeeper localhost:2181 --group test-consumer-group`
 - 输出结果：
 - Group Topic Pid Offset logSize Lag Owner test-consumer-group my-replicated-topic 0 0 0 0 none test-consumer-group my-replicated-topic 1 3 4 1 none test-consumer-group my-replicated-topic 2 0 0 0 none BROKER INFO 0 -> saurabh-Inspiron:9092
 - 参数详解：
 - group: This accepts the name of the consumer group of which the offsets are monitored.
 - zookeeper: This accepts comma-separated values for ZooKeeper in the host:offsetformat.
 - topic: This accepts the topic name in a comma-separated format such as topic1, topic2,topic3. It will help in monitoring multiple topics.
 - broker-info: If this parameter is set, it will print the broker details for the topic as well.
 - help: This prints the help message that includes the details of all the parameters.
 - 子主题 4
 - 2、理解转储日志片段
 - `$ bin/kafka-run-class.sh kafka.tools.DumpLogSegments --deep-iteration --files /tmp/kafka-logs/my-replicated-topic-2/00000000000000000000.log`
 - 输出结果：
 - Dumping /tmp/kafka-logs/my-replicated-topic-2/00000000000000000000.log
 - Starting offset: 0
 - offset: 0 position: 0 invalid: true payloadsize: 4 magic: 0 compresscodec: NoCompressionCodec crc: 1495943047
 - offset: 1 position: 30 invalid: true payloadsize: 7 magic: 0 compresscodec: NoCompressionCodec crc: 2252241273
 - offset: 2 position: 63 invalid: true payloadsize: 7 magic: 0 compresscodec: NoCompressionCodec crc: 2511132036
 - offset: 3 position: 96 invalid: true payloadsize: 11 magic: 0 compresscodec: NoCompressionCodec crc: 4090103826
 - offset: 4 position: 133 invalid: true payloadsize: 6 magic: 0 compresscodec: NoCompressionCodec crc: 3891823159
 - offset: 5 position: 165 invalid: true payloadsize: 4 magic: 0 compresscodec: NoCompressionCodec crc: 2440616224
 - 参数详解：

- `--deep-iteration`: If set, it uses deep instead of shallow iteration to examine the log files.
 - `--files`: This is the only mandatory field. It is a comma-separated list of data and index log files that need to be dumped.
 - `--max-message-size`: This is used to set the size of the largest message. The default value of this is 5242880.
 - `--print-data-log`: This parameter must be set if you want the messages' content while dumping the data logs.
 - `--verify-index-only`: This parameter must be set if you want to verify the index log without printing its content.
- 3、导出 Zookeeper Offsets
- 在 Kafka 文件夹下执行如下命令：
 - `$ bin/kafka-run-class.sh kafka.tools.ExportZkOffsets --zkconnect localhost:2181 --group test-consumer-group --output-file /tmp/out.txt`
 - `$ bin/kafka-run-class.sh kafka.tools.ExportZkOffsets --zkconnect vlionserver-216:2181,vlionserver-219:2181,vlionserver-221:2181/kafka --group test-consumer-group --output-file /tmp/out.txt`
 - `$ cat /tmp/out.txt`
 - `/consumers/test-consumer-group/offsets/mytesttopic/3:0 /consumers/test-consumer-group/offsets/mytesttopic/2:6 /consumers/test-consumer-group/offsets/mytesttopic/1:0 /consumers/test-consumer-group/offsets/mytesttopic/0:0`
 - 参数的详解：
 - `--zkconnect`: This specifies the ZooKeeper connection string. It will be comma-separated in the host:port format.
 - `--group groupname`: This specifies the consumer's group name.
 - `--help`: This prints the help message.
 - `--output-file`: This field specifies the file the ZooKeeper offsets should be written to.
- 4、导入 Zookeeper Offsets
- 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.ImportZkOffsets --inputfile /tmp/zkoffset.txt --zkconnect localhost:2181`
 - 参数详解：
 - `--zkconnect`: This specifies the ZooKeeper connect string. It will be comma-separated in the host:port format.
 - `--input-file`: This specifies the file to import ZooKeeper offsets from.
 - `--help`: This prints the help message.
- 5、理解 GetOffsetShell
- 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.GetOffsetShell --broker-list localhost:9092 --topic mytesttopic --time -1`
 - `mytesttopic:0:0 mytesttopic:1:0 mytesttopic:2:6 mytesttopic:3:0`
 - 参数详解：
 - `--broker-list`: This specifies the list of server ports to connect to. This can be a list of servers in the host:port format. You can specify more than one by comma-separating them.
 - `--max-wait-ms`: This specifies the maximum amount of time each of the fetch requests will wait. The default value for this is 1000, that is, 1 second.
 - `--offsets`: This specifies the number of offsets that are returned. By default, it will return only one offset.
 - `--partitions`: This is a comma-separated list of partition IDs. If it is not specified, it will fetch the offsets for all the partitions by default.
 - `--time`: This specifies the timestamp to fetch for the offsets. -1 is for the latest and -2 for the earliest.
 - `--topic`: This specifies the topic for which the offset needs to be fetched.
- 6、使用 JMX 工具
- 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.JmxTool --jmx-url service:jmx:rmi:///jndi/rmi://127.0.0.1:9999/jmxrmi`
 - 参数详解：
 - `--attributes`: This is a comma-separated list of objects that acts as a whitelist of attributes to be queried. All the objects are reported if none are mentioned.
 - `--date-format`: This specifies the data format to be used for the time field. Please refer to `java.text.SimpleDateFormat` for all the available options.
 - `--help`: This prints the help message.
 - `--jmx-url`: This specifies the URL to connect to the poll JMX data. See Oracle javadocor `JMXServiceURL` for details.

- `--object-name`: This specifies the JMX object name to be used as a query. This can contain wild cards. This option can be given multiple times to specify more than one query. If no objects are specified, all the objects will be queried.
 - `--reporting-interval`: This specifies the interval in milliseconds with the poll JMX stats. The default reporting interval is 2 seconds.
- 7、使用 Kafka 迁移工具
 - 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.KafkaMigrationTool --consumer.config consumer.config --kafka.0.07.jar /opt/kafka7/bin/kafka.jar --num.producer 4 --num.streams 4 --producer.config producer.config --zkclient.01.jar /opt/kafka7/bin/zkclient.jar`
 - 参数详解：
 - `--blacklist`: This contains a list of the topics that are blacklisted from being migrated from the 0.7 cluster.
 - `--consumer.config`: This specifies the path to the Kafka 0.7 consumer configuration file to consume from the source 0.7 cluster. Multiple of these might be specified.
 - `--help`: This prints the help message.
 - `--kafka.07.jar`: This specifies the path to the Kafka 0.7 jar file.
 - `--num.producers`: This specifies the number of producer instances. The default value, if not specified, is 1.
 - `--num.streams`: This specifies the number of consumer streams. The default value, if not specified, is 1.
 - `--producer.config`: This specifies the path to the Kafka producer configuration file.
 - `--queue.size`: This specifies the queue size in the number of messages that are buffered between the 0.7 consumer and the 0.8 producer. The default value for this is 10000.
 - `--whitelist`: This specifies the whitelist of topics to be migrated from the 0.7 cluster.
 - `--zkclient.01.jar`: This specifies the path to the zkClient 0.1.jar file required by Kafka 0.7.
- 8、MirrorMaker 工具
 - 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.MirrorMaker --consumer.config config/consumer.config --producer.config config/producer.config --whitelist mytesttopic`
 - 参数详解：
 - `--blacklist`: This specifies the blacklist of topics to be mirrored. This can be a regular expression as well.
 - `--consumer.config`: This specifies the path to the consumer configuration file to consume from a source cluster. Multiple files may be specified.
 - `--help`: This prints the help message.
 - `--num.producers`: This specifies the number of producer instances. By default, only one producer instance will be created.
 - `--num.streams`: This specifies the number of consumption streams' threads. By default, only one thread will be started.
 - `--producer.config`: This specifies the path to the embedded producer configuration file.
 - `--queue.size`: This specifies the queue size in the number of messages that are buffered terms between the consumer and producer. The default value for it is 10000.
 - `--whitelist`: This specifies the whitelist of topics to be mirrored.
- 9、日志 Producer 的回放 (Replay)
 - 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.ReplayLogProducer --sync --broker-list localhost:9092 --inputtopic mytesttopic --outputtopic mytesttopic2 --zookeeper localhost:2181`
 - 参数详解：
 - `--sync`: If this is specified, the messages are sent synchronously; else they are sent asynchronously.
 - `--broker-list`: This specifies the broker list. This is a mandatory field.
 - `--inputtopic`: This specifies the topic to consume from.
 - `--messages`: This specifies the number of messages to be sent. Its default value is -1 that means infinite.
 - `--outputtopic`: This specifies the topic to produce to.
 - `--reporting-interval`: This specifies the interval in milliseconds to print the progress information. Its default value is 5000.
 - `--threads`: This specifies the number of sending threads. By default, only one thread is used.
 - `--zookeeper`: This specifies the connection string for the zookeeper connection in the host:port form. Multiple URLs can be given to allow fail over.
- 10、简单的 Consumer Shell
 - 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.SimpleConsumerShell --broker-list localhost:9092 --max-messages 10 --offset -2 --partition 0 --print-offsets --topic mytesttopic`

- 参数详解：
 - --broker-list: This specifies the list of server ports to connect to. This can be a list of servers in the host:port format. You can specify more than one by comma-separating them.
 - --clientId: This specifies the ID of the client. By default, it is SimpleConsumerShell.
 - --fetchsize: This specifies the size of each fetch request. The default value for this is 1048576.
 - --formatter: Using this, you can specify the name of the class to be used to format Kafka messages using something other than kafka.consumer.DefaultMessageFormatter.
 - --property: This specifies the arguments for the formatter.
 - --max-messages: This specifies the number of messages to be consumed. By default, it consumes 2147483647 messages.
 - --max-wait-ms: This specifies the maximum amount of time each fetch request will wait in milliseconds. By default, it will wait for 1 second.
 - --no-wait-at-logend: If this is specified, then the consumer will stop on reaching the end of the log and not wait for new messages.
 - --offset: This specifies the ID to consume from. The default value for this is -2, which means from the beginning. If -1 is specified, it means the end.
 - --partition: This specifies the partition to read from. If it is not specified, it will read from partition 0.
 - --print-offsets: If this is specified, the offset needs to be printed as well.
 - --replica: This specifies the replica ID to consume from. Its default value is -1, which means read from the lead broker.
 - --skip-message-on-error: If this is specified, then it will skip the message in case of an error instead of halting.
 - --topic: This specifies the topic to consume from.
- 11、将状态改变的日志进行合并
 - 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.StateChangeLogMerger --log-regex /tmp/state-change.log* --partitions 0,1,2 --topic statelog`
 - 参数详解：
 - --end-time: This specifies the latest timestamp of state change entries to be merged in the java.text.SimpleDateFormat format. If it is not specified, the default value will be 9999-12-31 23:59:59,999.
 - --logs: This is used to specify a comma-separated list of state change logs or regex for the log file names. Either this or the logs-regex parameter can be used at one time.
 - --logs-regex: This is used to specify a regex to match the state change log files to be merged. Either this or the logs parameter can be used at one time.
 - --partitions: This specifies a comma-separated list of partition IDs whose state change logs should be merged.
 - --start-time: This specifies the earliest timestamp of state change entries to be merged in the java.text.SimpleDateFormat format. If it is not specified, the default value will be 0000-00-00 00:00:00,000.
 - --topic: This specifies the topic whose state change logs should be merged.
- 12、在 Zookeeper 中更新 Offsets
 - 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.UpdateOffsetsInZK earliest config/consumer.properties mytopic`
- 13、检验 Consumer 的 Rebalance
 - 查看 Kafka 的分区信息：
 - `/consumers/[consumer_group]/owners/[topic]/[broker_id-partition_id]`
 - 执行命令：
 - `$ bin/kafka-run-class.sh kafka.tools.VerifyConsumerRebalance --zookeeper.connect localhost:2181 --group mytestconsumer`
 - 参数详解：
 - --group: This is used to specify the consumer group.
 - --help: This prints the help message.
 - --zookeeper.connect: This is used to specify the ZooKeeper connect string.
- C5 - Kafka 与 Java 的整合
 - 1、写一个简单的 Producer
 - 2、写一个简单的 Consumer
 - 3、写一个 High level 的 consumer, 多线程场景：
 - 4、利用消息的 Partition 来写个 Producer (Code + Project)

How it works...


We start with a general `KafkaProducer` class object, with the same properties we used for a simple producer. All the steps are pretty much a standard routine. When creating a `ProducerRecord` for the message, we need to pass the following parameters: topic name, partition number, message key, and the actual message. The partition number is the partition to which you want to send the data. You should be careful while selecting the partition number because, if chosen incorrectly, it might slow down your data ingestion and affect the performance of your system. It is highly recommended that you distribute the load evenly across partitions. You can, after creating this record, call the method `send` on the producer object and pass this record as a parameter. After you have sent all the messages it is important that you call `close` on the producer object.

There's more...

In the Kafka producer, a partition key can be specified to indicate the destination partition of the message. By default, a hashing-based partitioner is used to determine the partition ID given the key, and people can use customized partitioners also. In Kafka 0.8 onwards, if no key is specified, Kafka will send it to a random partition. It will then keep on sending it to that particular topic unless the metadata for the topic is refreshed, which by default is every 10 minutes. This might lead to issues where a large number of partitions are unused when there are fewer producers than partitions. So it is recommended that you use a message key and a custom random partitioner or reduce the metadata refresh interval.

• 项目代码:

- `Producer with partitioning.zip`

 `Producer with partit ...`

• i) Kafka producer 的属性配置

- `Properties properties = new Properties();`
- `properties.put("metadata.broker.list", "127.0.0.1:9092");`
- `properties.put("serializer.class", "kafka.serializer.StringEncoder");`
- `properties.put("partitioner.class", "com.kafkacookbook.SimplePartitioner");`

- 指定分区的属性

- `Producer with partitioning.zip`



- `properties.put("request.required.acks", "1");`

• ii) 在 Java 中利用配置的属性来实例化一个 `Producer` 对象

- `KafkaProducer<Integer, String> producer = new KafkaProducer<Integer, String>(properties);`

• iii) 向指定的 Partition 发送100个不同的消息:

- `for(int iCount = 0; iCount < 100; iCount++){`
- `int partition = iCount % producer.partitionsFor("mytesttopic").size();`
- `String message = "My Test Message No " + iCount;`
- `ProducerRecord<Integer,String> record = new ProducerRecord<Integer, String>("mytesttopic",`
- `partition, iCount, message);`
- `producer.send(record);`
- `}`

• iv) 关闭 producer 实例:

- `producer.close();`

• 5、使用多线程来消费 Kafka (Code + Project)

• 项目代码:

- `MultiThreadedConsumer.zip`



• i) 实现 `Runnable` 类, 并且定义 `run()` 方法:

```
1 package com.kafkacookbook;
2
3 import kafka.consumer.ConsumerIterator;
4 import kafka.consumer.KafkaStream;
5
6 /**
7  * Created by saurabh on 4/8/15.
8  */
9 class ConsumerThread implements Runnable {
10     private KafkaStream stream;
11     private Integer threadNumber;
12
13     public ConsumerThread(KafkaStream stream, int threadNumber) {
14         this.stream = stream;
15         this.threadNumber = threadNumber;
16     }
17
18     public void run() {
19         ConsumerIterator<byte[], byte[]> it = stream.iterator();
20         while (it.hasNext()) {
21             System.out.println("Message from thread " + threadNumber + ": " + new String(it.next().message()));
22         }
23         System.out.println("Shutting down thread: " + threadNumber);
24     }
25 }
26
27
28
```

实现一个 `Runnable` 的类, 并且将 `KafkaStream` 和 `threadNumber` 作为参数字段

实现 `run` 方法

• ii) 创建一个 `ConsumerConnection` 来加载 Kafka 的自定义属性:

- `Properties properties = new Properties();`

- `properties.put("zookeeper.connect", "localhost:2181");`
 - `properties.put("group.id", "mygroup");`
 - `properties.put("zookeeper.session.timeout.ms", "500");`
 - `properties.put("zookeeper.sync.time.ms", "250");`
 - `properties.put("auto.commit.interval.ms", "1000");`
 - `ConsumerConnector consumer = Consumer.createJavaConsumerConnector(new ConsumerConfig(properties));`
- iii) 创建一个 Map 来存放 topic 和 stream 的个数:
 - `Map<String, Integer> topicCount = new HashMap<String, Integer>();`
 - `topicCount.put("mytesttopic", 4);`
- iv) 根据消息的配置获取 consumer 的数据流:
 - `Map<String, List<KafkaStream<byte[], byte[]>>> consumerStreams = consumer.createMessageStreams(topicCount);`
 - `List<KafkaStream<byte[], byte[]>> streams = consumerStreams.get("mytesttopic");`
- v) 使用递归来读取数据流:
 - `ExecutorService executor = Executors.newFixedThreadPool(4);`
 - `int threadNumber = 0;`
 - `for (final KafkaStream stream : streams) {`
 - `executor.submit(new ConsumerThread(stream, threadNumber));`
 - **调用 ConsumerThread**
 - `threadNumber++;`
 - `}`
- C6 - Kafka 的使用
 - 1、添加 & 删除 Topics
 - 创建一个 Topic
 - `> bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic testtopic --partitions 10 --replication-factor 2 --config max.message.bytes=64000`
 - 删除一个 Topic
 - `> bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic testtopic`
 - 涉及参数详解:
 - `--zookeeper`: This specifies the ZooKeeper connect string; it can be comma-separated in the format host:port.
 - `--create`: This keyword specifies that a topic needs to be created.
 - `--delete`: This keyword specifies that the topic needs to be deleted. The server configuration has to be `delete.topic.enable=true`. By default this is set as false. If it is false, the topic will never be deleted.
 - `--topic`: This is used to specify the topic name. The topic name must follow this keyword.
 - `--partitions`: This is used to specify the number of partitions to be created for the topic. The number of partitions to be created must follow this keyword.
 - `--replication-factor`: This specifies the number of replicas to be created for the topic. This number must be less than the number of nodes in the cluster.
 - Other configurations needed for the topic can be specified by using the following format: `--config x=y`, where x is config name and y is the value for the configuration. These are used to override the default property set on the server.
 - Details of the various configurations are as follows:
 - `cleanup.policy`: This keyword can take either of two values: delete or compaction. The default value is delete, which will delete the logs once the logs reach the time or size limits.
 - `delete.retention.ms`: This is used to change the length of time you want the logs to be retained in Kafka.
 - 2、修改 Topics
 - 执行命令:
 - `bin/kafka-topics.sh --zookeeper localhost:2181/chroot --alter --topic my_topic_name --partitions 40 --config delete.retention.ms=10000 --deleteConfig retention.ms`
 - 参数详解:
 - `--zookeeper`: This keyword is used to mention the ZooKeeper host and port number for the cluster along with the optional path to be used in ZooKeeper. This is useful if you have more than one Kafka cluster using the same ZooKeeper cluster.
 - `--topic`: This keyword is followed by the name of the topic that needs to be modified.
 - `--partitions`: This keyword is followed by the number of partitions to be set for the topic specified.
 - `--config`: This keyword, if followed by a config in the format configname=value, sets the new value given.

- `--deleteConfig`: This keyword followed by a config name removes the config from the topic.
- 3、实现一个委婉的/优雅的 关闭
 - 修改配置文件：
 - `controlled.shutdown.enable=true`
 - 执行命令：
 - `> bin/kafka-server-stop.sh`
- 4、平衡 Leadership
 - 执行命令：
 - `> bin/kafka-preferred-replica-election.sh --zookeeper localhost:2181/chroot`
 - `auto.leader.rebalance.enable=true` (可选项)
 - 参数详解：
 - `--zookeeper`: This keyword is used to mention the ZooKeeper host and port number for the clusters along with the optional path to be used in ZooKeeper. This is useful if you have more than one Kafka cluster using the same ZooKeeper cluster.
- 5、在 Kafka 集群之间进行数据镜像
 - 执行命令：
 - `> bin/kafka-run-class.sh kafka.tools.MirrorMaker --consumer.config consumer.properties --producer.config producer.properties --whitelist testtopic`
 - 参数详解：
 - `--consumer.config`: This keyword is followed by the consumer properties file for the cluster being mirrored. You can provide more than one consumer config to mirror topics from different clusters in one go.
 - `--producer.config`: This keyword is followed by the producer properties file for the cluster being mirrored to.
 - `--whitelist`: This follows the regular expression for the topics that need to be copied from the cluster mentioned. Giving this as `*` will move all topics in the cluster being mirrored to the destination cluster.
 - `--blacklist`: This follows the regular expression for the topics that need not be mirrored from the cluster to be mirrored.
- 6、集群的扩展
 - 创建一个 JSON 文件 (topics-to-move.json)
 - `{"topics": [{"topic": "foo1"},`
 - `{"topic": "foo2"}],`
 - `"version":1`
 - `}`
 - 执行如下命令：
 - `> bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --topics-to-move-json-file topics-to-move.json --broker-list "5,6" --generate`
 - `Current partition replica assignment`
 - 运行结果：
 - `{"version":1,`
 - `"partitions":[{"topic":"foo1","partition":2,"replicas":[1,2]},`
 - `{"topic":"foo1","partition":0,"replicas":[3,4]},`
 - `{"topic":"foo2","partition":2,"replicas":[1,2]},`
 - `{"topic":"foo2","partition":0,"replicas":[3,4]},`
 - `{"topic":"foo1","partition":1,"replicas":[2,3]},`
 - `{"topic":"foo2","partition":1,"replicas":[2,3]}`
 - `}`
 - `Proposed partition reassignment configuration`
 - `["version":1,`
 - `"partitions":[{"topic":"foo1","partition":2,"replicas":[5,6]},`
 - `{"topic":"foo1","partition":0,"replicas":[5,6]},`
 - `{"topic":"foo2","partition":2,"replicas":[5,6]},`
 - `{"topic":"foo2","partition":0,"replicas":[5,6]},`
 - `{"topic":"foo1","partition":1,"replicas":[5,6]},`
 - `{"topic":"foo2","partition":1,"replicas":[5,6]}]`
 - `}`

- 830 上测试截图：

```

vlonserver@vlonserver:830 bin$ sh kafka-reassign-partitions.sh --zookeeper vlonserver:830:2181/kafka_test --topics-to-move-json-file topics-to-move.json --broker-list "5,6" --generate
Current partition replica assignment

{"version":1,"partitions":[{"topic":"test4_DPTopic","partition":0,"replicas":[0]},{"topic":"test4_VlionDSPLog","partition":6,"replicas":[0]},{"topic":"test4_DPTopic","partition":4,"replicas":[0]},{"topic":"test4_VlionDSPLog","partition":1,"replicas":[0]},{"topic":"test4_DPTopic","partition":7,"replicas":[0]},{"topic":"test4_VlionDSPLog","partition":8,"replicas":[0]},{"topic":"test4_DPTopic","partition":3,"replicas":[0]},{"topic":"test4_VlionDSPLog","partition":5,"replicas":[0]},{"topic":"test4_DPTopic","partition":1,"replicas":[0]},{"topic":"test4_VlionDSPLog","partition":2,"replicas":[0]},{"topic":"test4_DPTopic","partition":9,"replicas":[0]},{"topic":"test4_VlionDSPLog","partition":9,"replicas":[0]}]}
Proposed partition reassignment configuration

{"version":1,"partitions":[{"topic":"test4_DPTopic","partition":0,"replicas":[5]},{"topic":"test4_DPTopic","partition":4,"replicas":[6]},{"topic":"test4_VlionDSPLog","partition":6,"replicas":[5]},{"topic":"test4_VlionDSPLog","partition":1,"replicas":[6]},{"topic":"test4_DPTopic","partition":7,"replicas":[5]},{"topic":"test4_VlionDSPLog","partition":8,"replicas":[5]},{"topic":"test4_DPTopic","partition":3,"replicas":[6]},{"topic":"test4_VlionDSPLog","partition":5,"replicas":[6]},{"topic":"test4_DPTopic","partition":1,"replicas":[5]},{"topic":"test4_VlionDSPLog","partition":2,"replicas":[6]},{"topic":"test4_DPTopic","partition":9,"replicas":[6]},{"topic":"test4_VlionDSPLog","partition":9,"replicas":[6]}]}
vlonserver@vlonserver:830 bin$

```

• 创建一个 JSON 文件，用来将指定的 Partition 移到指定的 Node

- custom-reassignment.json

- {"version":1,
- "partitions":[{"topic":"foo1","partition":2,"replicas":[3,6]},
- {"topic":"foo1","partition":0,"replicas":[4,6]},
- {"topic":"foo2","partition":2,"replicas":[5,6]},
- {"topic":"foo2","partition":0,"replicas":[3,6]},
- {"topic":"foo1","partition":1,"replicas":[4,6]},
- {"topic":"foo2","partition":1,"replicas":[5,6]}]
- }

- 执行命令：

- > bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file custom-reassignment.json --execute
- bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file custom-reassignment.json --verify

- 参数详解：

- --zookeeper: This specifies the ZooKeeper connect string; it can be comma-separated in the format host:port.
- --topics-to-move-json-file: This specifies the path to the JSON file that we created in the previous step.
- --broker-list: This takes in the list of brokers to assign the new replicas to. The broker IDs are given in a comma-separated format as 1,2,3. This is required if the topic-to-move-json-file parameter is mentioned.
- --generate: This argument tells the tools to generate a new candidate configuration for the topics. This will not actually start the migration of replicas.

• 7、增加复制因子

- 创建 JSON 文件 increase-replication-factor.json：

- {"version":1,
- "partitions":[{"topic":"mytesttopic","partition":0,"replicas":[5,6,7]}]}

- 执行命令：

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

- 检验分区分配的情况：

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

• 8、检查 Consumer 的位置

- 执行命令：

- > bin/kafka-run-class.sh kafka.tools.ConsumerOffsetChecker --zkconnect localhost:2181 --group my-test-group
- Vlion 线上：
- sh kafka-run-class.sh kafka.tools.ConsumerOffsetChecker --zkconnect vlonserver-216:2181,vlonserver-219:2181,vlonserver-221:2181/kafka --group realtime_dsp_Report_0620_01

- 返回结果：

- Group Topic Pid Offset logSize Lag Owner
- my-test-group mytesttopic 0 0 0 0 test_sminni-er-1638490550254-85375431-0
- my-test-group mytesttopic 1 0 0 0 test_sminni-er-1638490550254-34523456-0

- 参数详解：

- --zkconnect: This is followed by the host and port number in the format host:port for the ZooKeeper for the Kafka cluster
- --group: This is the consumer group ID you want to check the offsets for

• 9、使 Brokers 停止运行

- 创建 JSON 文件 change-replication-factor.json：

- {"version":1,
 - "partitions":[{"topic":"mytesttopic","partition":0,"replicas":[1,2]}]}
- 执行命令：
 - > bin/kafka-reassign-partitions.sh --zookeeper localhost:2181 --reassignment-json-file change-replication-factor.json --execute
- C7 - Kafka 与 第三方平台的整合
 - **Flume**
 - 配置文件: flume.conf
 - flume1.sources = kafka-source-1
 - flume1.channels = mem-channel-1
 - flume1.sinks = kafka-sink-1
 -
 - flume1.sources.kafka-source-1.type = org.apache.flume.source.kafka.KafkaSource
 - flume1.sources.kafka-source-1.zookeeperConnect = localhost:2181
 - flume1.sources.kafka-source-1.topic = srctopic
 - flume1.sources.kafka-source-1.batchSize = 100
 - flume1.sources.kafka-source-1.channels = mem-channel-1
 -
 - flume1.channels.mem-channel-1.type = memory
 -
 - flume1.sinks.kafka-sink-1.channel = mem-channel-1
 - flume1.sinks.kafka-sink-1.type = org.apache.flume.sink.kafka.KafkaSink
 - flume1.sinks.kafka-sink-1.batchSize = 50
 - flume1.sinks.kafka-sink-1.brokerList = localhost:9092
 - flume1.sinks.kafka-sink-1.topic = desttopic
 - 重启 Flume
 - > flume-ng agent --conf-file flume.conf --name flume1
 - **Gobblin**
 - 配置文件: mygobblin.conf
 - job.name=MyKafkaTest
 - job.group=MyTest
 - job.description=My sample Kafka Gobblin setup
 - job.lock.enabled=false
 -
 - source.class=gobblin.source.extractor.extract.kafka.KafkaAvroSource
 -
 - extract.namespace=gobblin.extract.kafka
 -
 - writer.destination.type=HDFS
 - writer.output.format=AVRO
 - writer.fs.uri=file://localhost/
 -
 - data.publisher.type=gobblin.publisher.TimePartitionedDataPublisher
 -
 - topic.whitelist=mytesttopic
 - bootstrap.with.offset=earliest
 -
 - kafka.brokers=localhost:2181
 -
 - writer.partition.level=hourly
 - writer.partition.pattern=YYYY/MM/dd/HH
 - writer.builder.class=gobblin.writer.AvroTimePartitionedWriterBuilder
 - writer.file.path.type=tablename
 - writer.partition.column.name=header.time
 -

- `mr.job.max.mappers=20`
-
- `extract.limit.enabled=true`
- `extract.limit.type=time`
- `extract.limit.time.limit=15`
- `extract.limit.time.limit.timeunit=minutes`
- 配置参数详解:
 - `job.name`: This specifies the name for the job.
 - `job.group`: This specifies the name for the job group.
 - `job.description`: This is used to give the description for the job.
 - `source.class`: This specifies the class to use as the source of your data. If you are using the AVRO file format and Kafka, you need to set it to `gobblin.source.extractor.extract.Kafka.KafkaAvroSource`. You can also make use of `gobblin.source.extractor.extract.Kafka.KafkaSimpleSources` if you are not using the AVRO file format. There are a bunch of other Source classes which you can use. They can be found in the github repo at <https://github.com/linkedin/gobblin/tree/master/gobblin-core/src/main/java/gobblin/source/extractor/extract>.
 - `extract.namespace`: This specifies the namespace for the extracted data. This namespace will be a part of default filename of the data written out.
 - `writer.destination.type`: This specifies the destination type for the writer task. Currently only HDFS is supported.
 - `writer.output.format`: This specifies the output format. At the time of writing, only the AVRO format is supported by Gobblin.
 - `writer.fs.uri`: This specifies the URI for the filesystem to write to.
 - `data.publisher.type`: This specifies the fully qualified name of the `DataPublisher` class that will publish the task data once everything has been completed.
 - `topic.whitelist`: This specifies a whitelist of topics from which data needs to be read.
 - `bootstrap.with.offset`: This tells Gobblin the offset from where it should start reading data from Kafka.
 - `Kafka.brokers`: This specifies the comma-separated Kafka brokers to ingest data from.
 - `writer.partition.level`: This specifies the partitioning level for the writer. The default value for this is daily.
 - `writer.partition.pattern`: This specifies the pattern in which the data written should be partitioned.
 - `writer.builder.class`: This is used to specify the class name of the writer builder.
 - `writer.file.path.type`: This is used to specify the file path type.
 - `writer.partition.column.name`: This specifies the column name of the partition.
 - `mr.job.max.mappers`: This is used to specify the number of tasks to launch. In MR mode, this will be the number of mappers launched. If the number of topic partitions to be pulled is larger than the number of tasks, the Kafka consumer will assign partitions to tasks in a balanced manner.
 - `extract.limit.enabled`: If this is set as true, then the task specifies a time limit.
 - `extract.limit.type`: This is used to specify the type of limit you want to set the task with. Its possible values are time, rate, count, and pool.
 - `extract.limit.time.limit`: This specifies the time limit on the tasks.
 - `extract.limit.time.limit.timeunit`: This is used to specify the unit of time to be used for the time limit.
- 单机模式下重启 Gobblin :
 - `gobblin-standalone.sh start --workdir gobblinworking --conf mygobblin.conf`
- Logstash
 - 配置文件: `logstash.conf`
 - `input {`
 - `kafka {`
 - `zk_connect => "localhost:2181"`
 - `topic_id => "mytesttopic"`
 - `consumer_id => "myconsumerid"`
 - `group_id => "mylogstash"`
 - `fetch_message_max_bytes => 1048576`
 - `}`
 - `}`
 - `output {`
 - `elasticsearch {`
 - `host => localhost`
 - `}`

- }
- 配置参数详解：
 - zk_connect: This specifies the ZooKeeper connect string; it can be comma-separated in the format host:port.
 - topic_id: This specifies the topic from which the source should read.
 - consumer_id: This specifies the consumer ID to be used while reading data from Kafka. It is automatically generated if not specified.
 - group_id: This specifies the group ID to be used by the Kafka consumer in Logstash. If not specified, its value by default is set to logstash.
 - fetch_message_max_bytes: This specifies the maximum number of bytes that might be fetched from Kafka for each topic-partition in each fetch request. This helps control the memory used by Logstash to store the message.
- 重启 Logstash
 - > bin/logstash -f logstash.conf
- **Spark**
 - 1、创建 SparkConf 和 Spark Streaming Context
 - SparkConf sparkConf = new SparkConf().setAppName("MySparkTest");
 - JavaStreamingContext jssc = new JavaStreamingContext(sparkConf, Durations.seconds(10));
 - 2、为 Topic 和 Kafka consumer 参数创建一个 HashSet
 - HashSet<String> topicsSet = new HashSet<String>();
 - topicsSet.add("mytesttopic");
 - HashMap<String, String> kafkaParams = new HashMap<String, String>();
 - kafkaParams.put("metadata.broker.list", "localhost:9092");
 - 使用逗号分隔符来分割主机 & 端口号的列表
 - 3、利用 Brokers 和 Topics 来创建 Kafka 数据流：
 - JavaPairInputDStream<String, String> messages = KafkaUtils.createDirectStream(
 - jssc,
 - String.class,
 - String.class,
 - StringDecoder.class,
 - StringDecoder.class,
 - kafkaParams,
 - topicsSet
 -);
- **Storm**
 - 1、利用 Zookeeper 地址来创建 ZkHosts 对象（格式 host:port）
 - BrokerHosts hosts = new ZkHosts("127.0.0.1:2181");
 - 2、实例化一个 SpoutConfig 配置实例，需要声明该对象的 scheme：
 - SpoutConfig kafkaConf = new SpoutConfig(hosts, "mytesttopic", "/brokers", "mytest");
 - kafkaConf.scheme = new SchemeAsMultiScheme(new StringScheme());
 - 3、利用 Kafka 的配置来实例化 KafkaSpout 对象：
 - KafkaSpout kafkaSpout = new KafkaSpout(kafkaConf);
 - 4、在 Topology 中调用该 Spout 实例：
 - TopologyBuilder builder = new TopologyBuilder();
 - builder.setSpout("spout", kafkaSpout, 10);
- **Elasticsearch**
 - 1、为 Elasticsearch 安装 Kafka 的数据流插件：
 - > bin/plugin -install kafka-river -url https://github.com/mariamhakobyan/elasticsearch-river-kafka/releases/download/v1.2.1/elasticsearch-river-kafka-1.2.1-plugin.zip
 - 2、运行如下的 curl 命令：
 - > curl -XPUT 'localhost:9200/_river/kafka-river/_meta' -d '{"type": "kafka", "kafka" : {"zookeeper.connect" : "localhost:2181", "zookeeper.connection.timeout.ms" : 10000, "topic" : "mytesttopic", "message.type" : "string"}, "index" : {"index" : "kafka-index", "type" : "status", "bulk.size" : 100, "concurrent.requests" : 1, "action.type" : "index", "flush.interval" : "12h"}, "statsd" : {"host" : "localhost", "prefix" : "kafka.river", "port" : 8125, "log.interval" : 10}}'
 - 配置参数详解：
 - type: This has the value kafka. This is required and should not be changed for this plugin.
 - Kafka: This is a JSON object containing the Kafka settings.

- `zookeeper.connect`: This specifies the ZooKeeper host address.
 - `zookeeper.connection.timeout.ms`: This specifies the ZooKeeper timeout in milliseconds. Its default value is 1000 (1 second).
 - `topic`: This specifies the name of the topic from which the plugin will read data to push to Elasticsearch.
 - `message.type`: This specifies the Kafka message type for the plugin to insert. It can take two values: `json` and `string`. If the message from Kafka is a JSON string then each JSON property will be inserted in the Elasticsearch as an individual document property. If it is a string, it will be inserted into the Elasticsearch document as a value.
 - `index`: This contains the JSON object for index properties.
 - `(index ->)index`: This specifies the name of the Elasticsearch index that the messages be will inserted into.
 - `type`: This specifies the type of the Elasticsearch index that the messages will be inserted into.
 - `bulk.size`: This specifies the number of messages that will be bulk-inserted into Elasticsearch. Its default value is 100.
 - `concurrent.requests`: This specifies the number of concurrent requests that will be allowed for indexing. A value of 0 means no concurrent requests and a value of 1 means one concurrent request will be allowed.
 - `action.type`: This specifies how the messages coming in should be processed. Its default value is `index`, which means that it creates a document with the `value` field set based on the message. If the value for this is set to `delete`, then the document with the `ID` field set in the message is deleted. The value `raw.execute` means that the message is executed as a raw query.
 - `flush.interval`: This specifies the amount of time the plugin should wait before pushing any remaining messages in Elasticsearch, even though the `bulk.size` has not been reached. This value can be defined as `10h` for 10 hours, `10m` for 10 minutes, or `10s` for 10 seconds. Its default value is 12 hours.
 - `statsd`: This is the object used to set the statsd configuration for statistics reporting.
 - `host`: This specifies the hostname for the statsd server.
 - `port`: This specifies the port number for the statsd server.
 - `prefix`: This specifies the prefix to be used for all statsd metric keys.
 - `log.interval`: This specifies the interval of time in seconds after which metrics should be reported to the statsd server. Its default value is 10 seconds.
- SolrCloud
 - 配置 flume.conf 文件:
 - `flume1.sources = kafka-source-1`
 - `flume1.channels = mem-channel-1`
 - `flume1.sinks = solr-sink-1`
 -
 - `flume1.sources.kafka-source-1.type = org.apache.flume.source.kafka.KafkaSource`
 - `flume1.sources.kafka-source-1.zookeeperConnect = localhost:2181`
 - `flume1.sources.kafka-source-1.topic = srctopic`
 - `flume1.sources.kafka-source-1.batchSize = 100`
 - `flume1.sources.kafka-source-1.channels = mem-channel-1`
 -
 - `flume1.channels.mem-channel-1.type = memory`
 -
 - `flume1.sinks.solr-sink-1.channel = mem-channel-1`
 - `flume1.sinks.solr-sink-1.type = org.apache.flume.sink.solr.morphline.MorphlineSolrSink`
 - `flume1.sinks.solr-sink-1.batchSize = 100`
 - `flume1.sinks.solr-sink-1.batchDurationMillis = 1000`
 - `flume1.sinks.solr-sink-1.morphlineFile = /etc/flume-ng/conf/morphline.conf`
 - `flume1.sinks.solr-sink-1.morphlineId = morphline1`
 - 重新启动 Flume :
 - `> flume-ng agent --conf-file flume.conf --name flume1`
 - 配置参数详解:
 - `type` : For solr, the type is defined as `org.apache.flume.sink.solr.morphline.MorphlineSolrSink`
 - `batchSize`: This specifies the number of messages to processed in one go
 - `batchDurationMillis`: This specifies the time to wait till the messages are processed in a batch, if the number of messages to be processed crosses the batch size number
 - `morphlineFile`: This specifies the path to the morphline configuration file
 - `morphlineId`: This specifies the identifier for the morphline configuration file if the configuration file has multiple ones

- Kafka 的实时配置

- 对 Kafka Producer 来说:

- `properties.put("request.required.acks", "1");`
 - `properties.put("linger.ms", "5");`
 - `properties.put("batch.size", "10");`

- Consumer 阶段:

- `props.put("consumer.request.timeout.ms", 100);`

- C8 - Kafka 的监控

- 1、监控 Server 的状态

- 在启动 Kafka Server 的时候指定 JMV_PORT

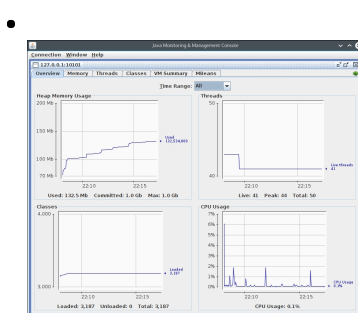
- `> JMX_PORT=10101 ./bin/kafka-server-start.sh config/server.properties`

- 使用 jconsole 来监控 Kafka 的状态:

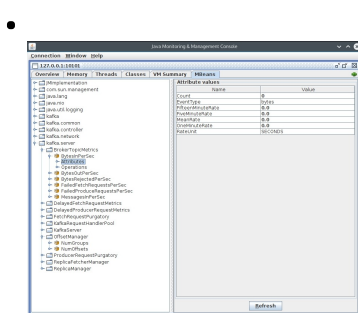
- `> jconsole 127.0.0.1:10101`

- 监控图:

- Overview



- MBeans

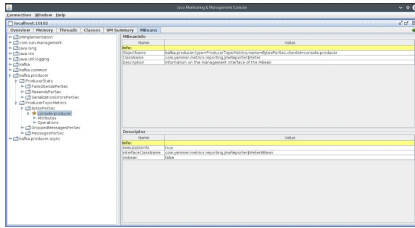


- 通过 JMX 端口可以查看 Kafka 所有的参数指标:

JConsole is the application that connects to the JMX port exposed by Kafka. You can read all the metrics from Kafka using JConsole. The details of the metrics with the MBean object name as exposed by Kafka are as follows:

- `kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec`: This gives the number of messages being inserted in Kafka per second. This has the attribute values given out as counts; one minute rate, five minute rate, fifteen minute rate, and mean rate.
- `kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions`: This specifies the number of partitions for which the number of replicas criterion is not met. If this value is anything more than zero, it means your cluster has issues replicating the partitions as desired by you.
- `kafka.controller:type=KafkaController,name=ActiveControllerCount`: This MBean gives the number of active controllers for Kafka for re-election.
- `kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs`: This MBean gives values of the rate at which leader election takes place as well as the latencies involved in that process. It gives latencies as mean 50th, 75th, 95th, 98th, 99th, and 99.9th latency percentiles. It also gives the time taken for leader election as a mean; one minute rate, five minute rate, and fifteen minute rate. It gives the count as well.
- `kafka.controller:type=ControllerStats,name=UncleanLeaderElectionsPerSec`: This gives unclean leader election statistics. It can give these values as a mean, one minute rate, five minute rate, and fifteen minute rate. It gives the count as well.
- `kafka.server:type=ReplicaManager,name=PartitionCount`: This MBean gives the total number of partitions present in that particular Kafka node.
- `kafka.server:type=ReplicaManager,name=LeaderCount`: This MBean gives the total number of leader partitions present in this Kafka node.
- `kafka.server:type=ReplicaManager,name=IsrShrinksPerSec`: This MBean specifies the rate at which in-sync replicas shrink. It can give these values as a mean, one minute rate, five minute rate, and fifteen minute rate. It gives the count of events as well.
- `kafka.server:type=ReplicaManager,name=IsrExpandsPerSec`: This MBean specifies the rate at which In-Sync replicas expand. It can give these values as a mean, one minute rate, five minute rate, and fifteen minute rate. It gives the count of events as well.

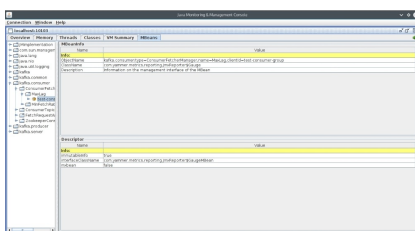
- `kafka.server:type=ReplicaFetcherManager,name=MaxLag,clientId=Replica`: This MBean specifies the maximum lag between the master and the replicas.
- 2、监控 Producer 状态
 - 在启动 Kafka producer 的时候指定 JMV_PORT
 - `>JMX_PORT=10102 bin/kafka-console-producer.sh --broker-list localhost:9092 --topic mytesttopic`
 - 通过 jconsole 来查看该端口的信息:



- 相关参数的详解:

If you switch to the MBeans tab in the jconsole app you can read various producer metrics, some of which are explained next. The `clientId` parameter is the producer client ID for which you want the statistics.

- `kafka.producer:type=ProducerRequestMetrics,name=ProducerRequestRateAndTimeMs,clientId=console-producer`: This MBean give values for the rate of producer requests taking place as well as latencies involved in that process. It gives latencies as a mean, the 50th, 75th, 95th, 98th, 99th, and 99.9th latency percentiles. It also gives the time taken to produce the data as a mean, one minute average, five minute average, and fifteen minute average. It gives the count as well.
 - `kafka.producer:type=ProducerRequestMetrics,name=ProducerRequestSize,clientId=console-producer`: This MBean gives the request size for the producer. It gives the count, mean, max, min, standard deviation, and the 50th, 75th, 95th, 98th, 99th, and 99.9th percentile of request sizes.
 - `kafka.producer:type=ProducerStats,name=FailedSendsPerSec,clientId=console-producer`: This gives the number of failed sends per second. It gives this value of counts, the mean rate, one minute average, five minute average, and fifteen minute average value for the failed requests per second.
 - `kafka.producer:type=ProducerStats,name=SerializationErrorsPerSec,clientId=console-producer`: This gives the number of serialization errors per second. It gives this value of counts, mean rate, one minute average, five minute average, and fifteen minute average value for the serialization errors per second.
 - `kafka.producer:type=ProducerTopicMetrics,name=MessagesPerSec,clientId=console-producer`: This gives the number of messages produced per second. It gives this value of counts, mean rate, one minute average, five minute average, and fifteen minute average for the messages produced per second.
- 3、监控 Consumer 状态
- 在启动 Kafka producer 的时候指定 JMV_PORT
 - `>JMX_PORT=10103 bin/kafka-console-producer.sh --broker-list localhost:9092 --topic mytesttopic`
 - 通过 jconsole 来查看该端口的信息:



- 相关参数详解:

If you switch to the MBeans tab in the jconsole app you can read various producer metrics, some of which are explained next. The parameter `clientId` is the consumer client id for which you want the statistics.

- `kafka.consumer:type=ConsumerFetcherManager,name=MaxLag,clientId=test-consumer-group`: This gives the number of messages that the consumer is behind by in consuming the messages pushed in by the producer.
- `kafka.consumer:type=ConsumerFetcherManager,name=MinFetchRate,clientId=test-consumer-group`: This gives the minimum rate at which the consumer sends fetch requests to the broker. If a consumer is dead, this value becomes close to 0.
- `kafka.consumer:type=ConsumerTopicMetrics,name=BytesPerSec,clientId=test-consumer-group`: This gives the number of bytes consumed per second. It gives this value of count, mean rate, one minute average, five minute average, and fifteen minute average for the bytes consumed per second.
- `kafka.consumer:type=ConsumerTopicMetrics,name=MessagesPerSec,clientId=test-consumer-group`: This gives the number of messages consumed per second. It gives this value of counts, mean rate, one minute average, five minute average, and fifteen minute average for the messages consumed per second.
- `kafka.consumer:type=FetchRequestAndResponseMetrics,name=FetchRequestRateAndTimeMs,clientId=test-consumer-group`: This MBean give values for the rate at which the consumer fetches the requests as well as the latencies involved in that process. It gives latencies as a mean, the 50th, 75th,

95th, 98th, 99th, and 99.9th latency percentiles. It also gives the time taken to consume the data as a mean, one minute rate, five minute rate, and fifteen minute rate. It gives the count as well.

- `kafka.consumer:type=FetchRequestAndResponseMetrics,name=FetchResponseSize,clientId=test-consumer-group`: This MBean gives the fetch size for the consumer. It gives the count, mean, max, min, standard deviation, 50th, 75th, 95th, 98th, 99th, and 99.9th percentile of request sizes.
- `kafka.consumer:type=ZookeeperConsumerConnector,name=FetchQueueSize,clientId=test-consumer-group,topic=mytesttopic,threadId=0`: This MBean gives the queue size for the fetch request for the clientId, threaded, and topic mentioned.
- `kafka.consumer:type=ZookeeperConsumerConnector,name=KafkaCommitsPerSec,clientId=test-consumer-group`: This MBean gives the fetch size for the Kafka commits per second. It gives the count, mean, one minute average, five minute average, and fifteen average rate of Kafka commits per second.
- `kafka.consumer:type=ZookeeperConsumerConnector,name=RebalanceRateAndTime,clientId=test-consumer-group`: This MBean gives the latency and rate of rebalance for the consumer. It gives latencies as a mean, the 50th, 75th, 95th, 98th, 99th, and 99.9th latency percentiles. It also gives the time taken to rebalance as a mean, one minute rate, five minute rate, and fifteen minute average. It also gives the count.
- `kafka.consumer:type=ZookeeperConsumerConnector,name=ZooKeeperCommitsPerSec,clientId=test-consumer-group`: This MBean gives the fetch size for the ZooKeeper commits per second. It gives the count, mean, one minute average, five minute average, and fifteen minute average rate of ZooKeeper commits per second.

• 4、连接到 Graphite



• 实现步骤:

- i) 从 Github 上下载 Kafka Graphite 的图形工具



- ii) 解压文件

- [unzip master.zip](#)

- iii) 在文件夹里运行如下命令:

- [mvn clean package](#)

- iv) 在 `libs/` 目录下添加两个 Jar 包:

Add `kafka-ganglia-1.0.0.jar` (located in the `./target` directory) and `metrics-ganglia-2.2.0.jar` (it should have been downloaded to the `./home/username/.m2/repository/com/yammer/metrics/metrics-ganglia/2.2.0` directory) to the `libs/` directory of your Kafka broker installation.

- [kafka-Graphite-1.0.0.jar](#)
- [metrics-Graphite-2.2.0.jar](#)

- v) 在 `server.properties` 配置文件中添加以下命令:

- [kafka.metrics.reporters=com.criteo.kafka.KafkaGraphiteMetricsReporter](#)
- [kafka.graphite.metrics.reporter.enabled=true](#)
- [kafka.graphite.metrics.host=localhost](#)
- [kafka.graphite.metrics.port=8649](#)
- [kafka.graphite.metrics.group=kafka](#)

- vi) 重启 Kafka , Graphite 报表系统将会接收从 Kafka 过来的数据

• 相关参数详解 (server.properties) :

- `kafka.metrics.reporters`: This tells Kafka the classes to load as Metrics Reporter. As mentioned in the first topic of this chapter, Kafka makes use of Yammer Metrics. You can have multiple metrics reports mentioned by comma-separating their classnames here. For Kafka Graphite Metrics Reporter, you need to mention it as `com.criteo.kafka.KafkaGraphiteMetricsReporter`.
- `kafka.Graphite.metrics.reporter.enabled`: This tells Kafka whether to enable Graphite Metrics. If the value for this is set as true, then metrics are reported. If it is set as false, metrics are not reported in Graphite.
- `kafka.Graphite.metrics.host`: This specifies the hostname for the Graphite system.
- `kafka.Graphite.metrics.port`: This specifies the port number of the Graphite system.
- `kafka.Graphite.metrics.group`: This specifies the group name that must be used to report metrics from this Kafka instance in Graphite.

• 5、使用 Ganglia 监控

• 实现步骤:

- i) 从 Github 上下载 Kafka Ganglia 的图形工具



- ii) 解压文件

- [unzip master.zip](#)

- iii) 在文件夹里运行如下命令:

- [mvn clean package](#)
- iv) 在 `libs/` 目录下添加两个 Jar 包：

Add `kafka-ganglia-1.0.0.jar` (located in the `./target` directory) and `metrics-ganglia-2.2.0.jar` (it should have been downloaded to the `home/username/.m2/repository/com/yammer/metrics/metrics-ganglia/2.2.0` directory) to the `libs/` directory of your Kafka broker installation.

 - [kafka-ganglia-1.0.0.jar](#)
 - [metrics-ganglia-2.2.0.jar](#)
- v) 在 `server.properties` 配置文件中添加以下命令：
 - [kafka.metrics.reporters=com.criteo.kafka.KafkaGangliaMetricsReporter](#)

 Kafka使用Yammer Metrics来报告服务端和客户端的Metric信息。Yammer Metrics 3.1.0提供6种形式的Metrics收集——Meters, Gauges, Counters, Histograms, Timers, Health Checks。与此同时, Yammer Metrics将Metric的收集与报告(或者说发布)分离, 可以根据需要自由组合。目前它支持的Reporter有Console Reporter, JMX Reporter, HTTP Reporter, CSV Reporter, SLF4J Reporter, Ganglia Reporter, Graphite Reporter。因此, Kafka也支持通过以上几种Reporter输出其Metrics信息。

 - [kafka.ganglia.metrics.reporter.enabled=true](#)
 - [kafka.ganglia.metrics.host=localhost](#)
 - 指定 Ganglia 系统的主机地址
 - [kafka.ganglia.metrics.port=8649](#)
 - 指定 Ganglia 系统的端口号
 - [kafka.ganglia.metrics.group=kafka](#)
 - 指定在 Ganglia 中维度报表使用的 Kafka 实例
- vi) 重启 Kafka , Ganglia 报表系统将会接收从 Kafka 过来的数据
- 相关参数详解 (server.properties) :
 - `kafka.metrics.reporters`: This tells Kafka the classes to load as Metrics Reporter. As mentioned in the first topic of this chapter, Kafka makes use of Yammer Metrics. You can have multiple metrics reports mentioned by comma-separating their classnames here. For Kafka Graphite Metrics Reporter, you need to mention it as `com.criteo.kafka.KafkaGangliaMetricsReporter`.
 - `kafka.ganglia.metrics.reporter.enabled`: This tells Kafka whether to enable Ganglia Metrics. If the value for this is set as true, then metrics are reported. If it is set as false, metrics are not reported in Ganglia.
 - `kafka.ganglia.metrics.host`: This specifies the hostname for the Ganglia system.
 - `kafka.ganglia.metrics.port`: This specifies the port number of the Ganglia system.
 - `kafka.ganglia.metrics.group`: This specifies the group name that must be used to report metrics from this Kafka instance in Ganglia.
- [kafka数据可靠性的深度解读 \(唯品会中间件团队\)](#)



• 消息中间件 (MQ) :

• Kafka

Kafka是一种高吞吐量的分布式发布订阅消息系统, 它可以处理消费者规模中的网站中的所有动作流数据。这种动作(网页浏览, 搜索和其他用户的行动)是在现代网络上的许多社会功能的一个关键因素。这些数据通常是由于吞吐量的要求而通过处理日志和日志聚合来解决。对于像Hadoop的一样的日志数据和离线分析系统, 但又要求实时处理的限制, 这是一个可行的解决方案。Kafka的目的是通过Hadoop的并行加载机制来统一线上和离线的消息处理, 也是为了通过集群机来提供实时的消费。

• 特点

- 通过O(1)的磁盘数据结构提供消息的持久化, 这种结构对于即使数以TB的消息存储也能够保持长时间的稳定性能。(文件追加的方式写入数据, 过期的数据定期删除)
- 高吞吐量: 即使是非常普通的硬件Kafka也可以支持每秒数百万的消息。
- 支持通过Kafka服务器和消费机集群来分区消息
- 支持Hadoop并行数据加载
- Kafka相关概念

• Kafka相关概念

Kafka相关概念

Broker

Kafka集群包含一个或多个服务器, 这种服务器被称为broker[5]

Topic

每条发布到Kafka集群的消息都有一个类别, 这个类别被称为Topic。(物理上不同Topic的消息分开存储, 逻辑上一个Topic的消息虽然保存于一个或多个broker上但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处)

Partition

Partition是物理上的概念, 每个Topic包含一个或多个Partition。

Producer

负责发布消息到Kafka broker

Consumer

消息消费者, 向Kafka broker读取消息的客户端。

Consumer Group

每个Consumer属于一个特定的Consumer Group (可为每个Consumer指定group name, 若不指定group name则属于默认的group)。

一般应用在大数据日志处理或对实时性(少量延迟), 可靠性(少量丢数据)要求稍低的场景使用。

- Broker
- Topic

- Partition
- Producer
- Consumer
- Consumer Group
- Kafka 的优化及在商业平台中的应用（搜狗商业广告平台）



• ZeroMQ

C 语言

- 特点
 - 高性能，非持久化；
 - 跨平台：支持Linux、Windows、OS X等
 - 多语言支持；C、C++、Java、.NET、Python等30多种开发语言
 - 可单独部署或集成到应用中使用

• ZeroMQ高性能设计要点

ZeroMQ高性能设计要点：

1、无锁的队列模型

对于跨线程间的交互（用户端和session）之间的数据交换通道pipe，采用无锁的队列算法CAS；在pipe两端注册有异步事件，在读或者写消息到pipe的时，会自动触发读写事件。

2、批量处理的算法

对于传统的消息处理，每个消息在发送和接收的时候，都需要系统的调用，这样对于大量的消息，系统的开销比较大，zeroMQ对于批量的消息，进行了适应性的优化，可以批量的接收和发送消息。

3、多核下的线程绑定，无须CPU切换

区别于传统的多线程并发模式，信号量或者临界区，zeroMQ充分利用多核的优势，每个核绑定运行一个工作者线程，避免多线程之间的CPU切换开销。

- 1、无锁的队列模型
- 2、批量处理的算法
- 3、多核下的线程绑定，无须CPU切换

• ActiveMQ :

- 特点
 - 1. 多种语言和协议编写客户端。语言: Java,C,C++,C#,Ruby,Perl,Python,PHP。应用协议: OpenWire,Stomp REST,WS Notification,XMPP,AMQP
 - 2. 完全支持JMS1.1和J2EE 1.4规范（持久化，XA消息，事务）
 - 3. 对spring的支持，ActiveMQ可以很容易内嵌到使用Spring的系统里面去，而且也支持Spring2.0的特性
 - 4. 通过了常见J2EE服务器（如 Geronimo,JBoss 4,GlassFish,WebLogic)的测试，其中通过JCA 1.5，resource adaptors的配置，可以让ActiveMQ可以自动的部署到任何兼容J2EE 1.4 商业服务器上
 - 5. 支持多种传送协议: in-VM,TCP,SSL,NIO,UDP,JGroups,JXTA
 - 6. 支持通过JDBC和journal提供高速的消息持久化
 - 7. 从设计上保证了高性能的集群，客户端-服务器，点对点
 - 8. 支持Ajax
 - 9. 支持与Axis的整合
 - 10. 可以很容易得调用内嵌JMS provider，进行测试

• RocketMQ - 阿里

• RocketMQ 系列文档



• 结构图

- 本地图解 ==>



• 执行流程

- (1) 客户端连接到消息队列服务器，打开一个channel
- (2) 客户端声明一个exchange，并设置相关属性
- (3) 客户端声明一个queue，并设置相关属性
- (4) 客户端使用routing key，在exchange和queue之间建立好绑定关系
- (5) 客户端投递消息到exchange
exchange接收到消息后，就根据消息的key和已经设置的binding，进行消息路由，将消息投递到一个或多个队列里

• Disruptor - 阿里面试



Disruptor 是英国外汇交易公司LMAX开发的一个高性能队列，研发的初衷是解决内存队列的延迟问题（在性能测试中发现竟然与I/O操作处于同样的数量级）。基于Disruptor开发的系统单线程能支撑每秒600万订单，2010年在QCon演讲后，获得了业界关注。2011年，企业应用软件专家Martin Fowler专门撰写长文介绍。同年它还获得了Oracle官方的Duke大奖。

从数据结构上来看，Disruptor是一个支持生产者->消费者模式的环形队列。能够在无锁的条件下进行并行消费，也可以根据消费者之间的依赖关系进行先后消费次序。本文将演示一些经典的场景如何通过Disruptor去实现。

目前，包括Apache Storm、Camel、Log4j 2在内的很多知名项目都应用了Disruptor以获取高性能

：美团技术详解 =>



- Java内置队列
- ArrayBlockingQueue的问题
 - 加锁
 - 关于锁和CAS
 - 锁
 - 原子变量
- 伪共享
 - 什么是共享
 - 缓存行
 - 什么是伪共享
- Disruptor的设计方案
 - Disruptor通过以下设计来解决队列速度慢的问题：
 - 环形数组结构
 - 为了避免垃圾回收，采用数组而非链表。同时，数组对处理器的缓存机制更加友好
 - 元素位置定位
 - 数组长度 2^n ，通过位运算，加快定位的速度。下标采取递增的形式。不用担心index溢出的问题。index是long类型，即使100万QPS的处理速度，也需要30万年才能用完
 - 无锁设计
 - 每个生产者或者消费者线程，会先申请可以操作的元素在数组中的位置，申请到之后，直接在该位置写入或者读取数据
 - 一个生产者
 - 写数据
 - 多个生产者
 - 读数据
 - 写数据
 - 总结
 - Disruptor通过精巧的无锁设计实现了在高并发情形下的高性能。
 - 在美团内部，很多高并发场景借鉴了Disruptor的设计，减少竞争的强度。其设计思想可以扩展到分布式场景，通过无锁设计，来提升服务性能。
 - 使用Disruptor比使用ArrayBlockingQueue略微复杂，为方便读者上手，增加代码样例。

• 消息队列设计 =>



• MQ 的通讯模式 =>

• 点对点通讯

点对点方式是最为传统和常见的通讯方式，它支持一对一、一对多、多对多、多对一等多种配置方式，支持树状、网状等多种拓扑结构

• 多点广播

MQ 适用于不同类型的应用。其中重要的，也是正在发展中的是“多点广播”应用，即能够将消息发送到多个目标站点 (Destination List)。可以使用一条 MQ 指令将单一消息发送到多个目标站点，并确保为每一站点可靠地提供信息。MQ 不仅提供了多点广播的功能，而且还拥有智能消息分发功能，在将一条消息发送到同一系统上的多个用户时，MQ 将消息的一个复制版本和该系统上接收者的名单发送到目标 MQ 系统。目标 MQ 系统在本地复制这些消息，并将它们发送到名单上的队列，从而尽可能减少网络的传输量。

• 发布/订阅 (Publish/Subscribe) 模式

发布/订阅功能使消息的分发可以突破目的队列地理指向的限制，使消息按照特定的主题甚至内容进行分发，用户或应用程序可以根据主题或内容接收到所需要的消息。发布/订阅功能使得发送者和接收者之间的耦合关系变得更为松散，发送者不必关心接收者的目的地址，而接收者也不必关心消息的发送地址，而只是根据消息的主题进行消息的收发。

• 群集 (Cluster)

为了简化点对点通讯模式中的系统配置，MQ 提供 Cluster(群集) 的解决方案。群集类似于一个域 (Domain)，群集内部的队列管理器之间通讯时，不需要两两之间建立消息通道，而是采用群集 (Cluster) 通道与其它成员通讯，从而大大简化了系统配置。此外，群集中的队列管理器之间能够自动进行负载均衡，当某一队列管理器出现故障时，其它队列管理器可以接管它的工作，从而大大提高系统的高可靠性。

• 队列高级特性设计：

• 消息的顺序

- 绝对的顺序消息基本上是不能实现的，当然在METAQ/Kafka等pull模型的消息队列中，单线程生产/消费，排除消息丢失，也是一种顺序消息的解决方案
- 条件 =>
 - 允许消息丢失。
 - 从发送方到服务方到接受者都是单点单线程。

• 消费确认

当broker把消息投递给消费者后，消费者可以立即响应我收到了这个消息。但收到了这个消息只是第一步，我能不能处理这个消息却不一定。或许因为消费能力的问题，系统的负荷已经不能处理这个消息；或者是刚才状态机里面提到的消息不是我想要接收的消息，主动要求重发。把消息的送达和消息的处理分开，这样才真正的实现了消息队列的本质-解耦。所以，允许消费者主动进行消费确认是必要的。当然，对于没有特殊逻辑的消息，默认Auto Ack也是可以的，但一定要允许消费方主动ack。对于正确消费ack的，没什么特殊的。但是对于reject和error，需要特别说明。reject这件事情，往往业务方是无法感知到的，系统的流量和健康状况的评估，以及处理能力的评估是一件非常复杂的事情。举个极端的例子，收到一个消息开始build索引，可能这个消息要处理半个小时，但消息量却是非常的小。所以reject这块建议做成滑动窗口/线程池类似的模型来控制，消费能力不匹配的时候，直接拒绝，过一段时间重发，减少业务的负担。但业务出错这件事情是只有业务方自己知道的，就像上文提到的状态机等等。这时应该允许业务方主动ack error，并可以与broker约定下次投递的时间。

- 重复消息是不可能100%避免的，除非可以允许丢失

● 消息重试

- 一般来讲，一个主流消息队列的设计范式里，应该是不丢消息的前提下，尽量减少重复消息，不保证消息的投递顺序。谈到重复消息，主要是两个话题：

- 问题 1 => 如何鉴别消息重复，并幂等的处理重复消息。

先来看看第一个话题，每一个消息应该有它的唯一身份。不管是业务方自定义的，还是根据IP/PID/时间戳生成的MessageId，如果有地方记录这个MessageId，消息到来是能够进行比对就能完成重复的鉴定。数据库的唯一键/bloom filter/分布式KV中的key，都是不错的选择。由于消息不能被永久存储，所以理论上都存在消息从持久化存储移除的瞬间上游还在投递的可能（上游因种种原因投递失败，不停重试，都到了下游清理消息的时间）。这种事情都是异常情况下才会发生的，毕竟是小众情况。两分钟消息都还没送达，多送一次又能怎样呢？幂等的处理消息是一门艺术，因为种种原因重复消息或者错乱的消息还是来到了

- 解决方案：

- 版本号

举个简单的例子，一个产品的状态有上线/下线状态。如果消息1是下线，消息2是上线。不巧消息1判重失败，被投递了两次，且第二次发生在2之后，如果不做重复性判断，显然最终状态是错误的。但是，如果每个消息自带一个版本号。上游发送的时候，标记消息1版本号是1，消息2版本号是2。如果再发送下线消息，则版本号标记为3。下游对于每次消息的处理，同时维护一个版本号。每次只接受比当前版本号大的消息。初始版本为0，当消息1到达时，将版本号更新为1。消息2到来时，因为版本号>1.可以接收，同时更新版本号为2。当另一条下线消息到来时，如果版本号是3.则是真实的下线消息。如果是1，则是重复投递的消息。如果业务方只关心消息重复不重复，那么问题就已经解决了。但很多时候另一个头疼的问题来了，就是消息顺序如果和想象的顺序不一致。比如应该的顺序是12，到来的顺序是21。则最后会发生状态错误。参考TCP/IP协议，如果想让乱序的消息最后能够正确的被组织，那么就应该只接收比当前版本号大一的消息。并且在一个session周期内要一直保存各个消息的版本号。如果到来的顺序是21，则先把2存起来，待1到来后，先处理1，再处理2，这样重复性和顺序性要求就都达到了。

- 状态机

基于版本号来处理重复和顺序消息听起来是个不错的主意，但凡事总有瑕疵。使用版本号的最大问题是：

1. 对发送方必须要求消息带业务版本号。

2. 下游必须存储消息的版本号，对于要严格保证顺序的。

还不能只存储最新的版本号的消息，要把乱序到来的消息都存储起来。而且必须对此做出处理。试想一个永不过期的“session”，比如一个物品的状态，会不停流转于上下线。那么中间环节的所有存储就必须保留，直到在某个版本号之前的版本一个不丢的到来，成本太高。就刚才的场景看，如果消息没有版本号，该怎么解决呢？业务方只需要自己维护一个状态机，定义各种状态的流转关系。例如，“下线”状态只允许接收“上线”消息，“上线”状态只能接收“下线消息”，如果上线收到上线消息，或者下线收到下线消息，在消息不丢失和上游业务正确的前提下。要么是消息发重了，要么是顺序到达反了。这时消费者只需要把“我不能处理这个消息”告诉投递者，要求投递者过一段时间重发即可。而且重发一定要有次数限制，比如5次，避免死循环，就解决了。举例子说明，假设产品本身状态是下线，1是上线消息，2是下线消息，3是上线消息，正常情况下，消息应该的到来顺序是123，但实际情况下收到的消息状态变成了3123。那么下游收到3消息的时候，判断状态机流转是下线->上线，可以接收消息。然后收到消息1，发现是上线->上线，拒绝接收，要求重发。然后收到消息2，状态是上线->下线，于是接收这个消息。此时无论重发的消息1或者3到来，还是可以接收。另外的重发，在一定次数拒绝后停止重发，业务正确。

- 问题 2 => 一个消息队列如何尽量减少重复消息的投递。

● 中间件对于重复消息的处理

回归到消息队列的话题来讲。上述通用的版本号/状态机/ID判重解决方案里，哪些是消息队列该做的、哪些是消息队列不该做业务方处理的呢？其实这里没有一个完全严格的定义，但回到我们的出发点，我们保证不丢失消息的情况下尽量少重复消息，消费顺序不保证。那么重复消息下和乱序消息下业务的正确，应该是由消费方保证的，我们要做的是减少消息发送的重复。我们无法定义业务方的业务版本号/状态机，如果API里强制需要指定版本号，则显得过于绑架客户了。况且，在消费方维护这么多状态，就涉及到一个消费方的消息落地/多机间的同步消费状态问题，复杂度指数级上升，而且只能解决部分问题

- 减少重复消息的关键步骤：

- broker记录MessageId，直到投递成功后清除，重复的ID到来不做处理，这样只要发送者在清除周期内能够感知到消息投递成功，就基本不会在server端产生重复消息。
- 对于server投递到consumer的消息，由于不确定对端是在处理过程中还是消息发送丢失的情况下，有必要记录下投递的IP地址。决定重发之前询问这个IP，消息处理成功了吗？如果询问无果，再重发

● 投递可靠性保证（最终一致性）

这是个激动人心的话题，完全不丢消息，究竟可不可能？答案是，完全可能，前提是消息可能会重复，并且，在异常情况下，要接受消息的延迟。方案说简单也简单，就是每当要发生不可靠的事情（RPC等）之前，先将消息落地，然后发送。当失败或者不知道成功失败（比如超时）时，消息状态是待发送，定时任务不停轮询有待发送消息，最终一定可以送达。具体来说：

对于各种不确定（超时、down机、消息没有送达、送达后数据没落地、数据落地了回复没收到），其实对于发送方来说，都是一件事情，就是消息没有送达。重推消息所面临的问题就是消息重复。重复和丢失就像两个噩梦，你必须面对一个。好在消息重复还有处理的机会，消息丢失再想找回就难了。Anyway，作为一个成熟的消息队列，应该尽量在各个环节减少重复投递的可能性，不能因为重复有解决方案就放纵的乱投递。最后说一句，不是所有的系统都要求最终一致性或者可靠投递，比如一个论坛系统、一个招聘系统。一个重复的简历或话题被发布，可能比丢失了一个发布显得更让用户无法接受。不断重复一句话，任何基础组件要服务于业务场景。

- producer往broker发送消息之前，需要做一次落地。

- 请求到server后，server确保数据落地后再告诉客户端发送成功。

- 支持广播的消息队列需要对每个待发送的endpoint，持久化一个发送状态，直到所有endpoint状态都OK才可删除消息。

● 消息持久化

● 支持不同消息模型

● 多实例集群功能

● 事务特性

分布式事务存在的最大问题是成本太高，两阶段提交协议，对于仲裁down机或者单点故障，几乎是一个无解的黑洞。对于交易密集型或者I/O密集型的应用，没有办法承受这么高的网络延迟，系统复杂性。并且成熟的分布式事务一定构建与比较靠谱的商用DB和商用中间件上，成本也太高。那如何使用本地事务解决分布式事务的问题呢？以本地和业务在一个数据库实例中建表为例子，与扣钱的业务操作同一个事务里，将消息插入本地数据库。如果消息入库失败，则业务回滚；如果消息入库成功，事务提交。然后发送消息（注意这里可以实时发送，不需要等定时任务检出，以提高消息实时性）。以后的问题就是前文的最终一致性问题所提到的了，只要消息没有发送成功，就一直靠定时任务重试。这里有一个关键的点，本地事务做的，是业务落地和消息落地的事务，而不是业务落地和RPC成功的事务。这里很多人容易混淆，如果是后者，无疑是事务嵌套RPC，是大忌，会有长事务死锁等各种风险。而消息只要成功落地，很大程度上就没有丢失的风险（磁盘物理损坏除外）。而消息只要投递到服务端确认后本地才做删除，就完成了producer->broker的可靠投递，并且当消息存储异常时，业务也是可以回滚的。本地事务存在两个最大的使用障碍：

1. 配置较为复杂，“绑架”业务方，必须本地数据库实例提供一个库表。

2. 对于消息延迟高敏感的业务不适用。

话说回来，不是每个业务都需要强事务的。扣钱和加钱需要事务保证，但下单和生成短信却不需要事务，不能因为要求发短信的消息存储投递失败而要求下单业务回滚。所以，一个完整的消息队列应该定义清楚自己可以投递的消息类型，如事务型消息，本地非持久型消息，以及服务端不落地的非可靠消息等。对不同的业务场景做不同的选择。另外事务的使用应该尽量低成本、透明化，可以依托于现有的成熟框架，如Spring的声明式事务做扩展。业务方只需要使用@Transactional标签即可。

- 两阶段提交，分布式事务。
- 本地事务，本地落地，补偿发送。

● 数据存储 ==>

● 基于 partition 的存储模型

● 优点 ==>

虽然我们并不想采用基于 partition 的存储模型，但是 Kafka 和 RocketMQ 里很多设计我们还是可以学习的：

- 顺序 append 文件，提供很好的性能
- 顺序消费文件，使用 offset 表示消费进度，不用给每个消息记录消费状态，成本极低
- 将所有 subject 的消息合并在一起，减少 partition 数量，单一集群可以支撑更多的 subject(RocketMQ)

● 添加一层拉取的 log(pull log) 来动态映射 consumer 与 partition 的逻辑关系

下图是 QMQ 的存储模型

先解释一下上图中的数字的意义。上图中方框上方的数字，表示该方框在自己 log 中的偏移，而方框内的数字是该项的内容。比如 message log 方框上方的数字:3,6,9 表示这几条消息在 message log 中的偏移。而 consume log 中方框内的数字 3,6,9,20 正对应着 message log 的偏移，表示这几个位置上的消息都是 topic1 的消息，consume log 方框上方的 1,2,3,4 表示这几个方框在 consume log 中的逻辑偏移。下面的 pull log 方框内的内容对应着 consume log 的逻辑偏移，而 pull log 方框外的数字表示 pull log 的逻辑偏移。

- 这样不仅解决了 consumer 的动态扩容缩容问题，还可以继续使用一个 offset 表示消费进度。而 pull log 与 consumer 一一对应。
- 这样存储中有三种重要的 log:
 - message log 所有 subject 的消息进入该 log，消息的主存储
 - consume log consume log 存储的是 message log 的索引信息
 - pull log 每个 consumer 拉取消息的时候会产生 pull log，pull log 记录的是拉取的消息在 consume log 中的 sequence
 - 那么消费者就可以使用 pull log 上的 sequence 来表示消费进度，这样一来我们就解耦了 consumer 与 partition 之间的耦合关系，两者可以任意的扩展。

● 延迟消息队列存储模型

● 概念 ==> (图)



除了对实时消息的支持，QMQ 还支持了任意时间的延时消息，在开源版本的 RocketMQ 里提供了多种固定延迟 level 的延时消息支持，也就是你可以发送几个固定的延时时间的延时消息，比如延时 10s, 30s...，但是基于我们现有的业务特征，我们觉得这种不同延时 level 的延时消息并不能满足我们的需要，我们更多的是需要任意时间延时。在 OTA 场景中，客人经常是预订未来某个时刻的酒店或者机票，这个时间是不固定的，我们无法使用几个固定的延时 level 来实现这个场景。

我们的延时 / 定时消息使用的是两层 hash wheel timer 来实现的。第一层位于磁盘上，每个小时为一个刻度，每个刻度会生成一个日志文件，根据业务特征，我们觉得支持两年内任意时间延时就够了，那么最多会生成 $2 * 366 * 24 = 17568$ 个文件。第二层在内存中，当消息的投递时间即将到来的时候，会将这个小时的消息索引（偏移量，投递时间等）从磁盘文件加载到内存中的 hash wheel timer 上。

● 在延时 / 定时消息里也存在三种 log:

- message log 和实时消息里的 message log 类似，收到消息后 append 到该 log，append 成功后就立即返回
- schedule log 按照投递时间组织，每个小时一个。该 log 是回放 message log 后根据延时时间放置对应的 log 上，这是上面描述的两层 hash wheel timer 的第一层，位于磁盘上
- dispatch log 延时 / 定时消息投递后写入，主要用于在应用重启后能够确定哪些消息已经投递

● 性能相关

● 同步 & 异步 ==>

● 异步 / 同步 / oneway

● 异步

- 归根结底你还是需要关心结果的，但可能不是当时的时间点关心，可以用轮询或者回调等方式处理结果

● 同步

- 是需要当时关心 的结果的

● oneway

- 是发出去就不管死活的方式，这种对于某些完全对可靠性没有要求的场景还是适用的，但不是我们重点讨论的范畴

● 客户端 & 服务端的 RPC 机制：

任何的RPC都是存在客户端异步与服务端异步的，而且是可以任意组合的：客户端同步对服务端异步，客户端异步对服务端异步，客户端同步对服务端同步，客户端异步对服务端同步。对于客户端来说，同步与异步主要是拿到一个Result，还是Future(Listenable)的区别。实现方式可以是线程池，NIO或者其他事件机制，这里先不展开讲。服务端异步可能稍微难理解一点，这个是需要RPC协议支持的。参考servlet 3.0规范，服务端可以吐一个future给客户端，并且在future done的时候通知客户端。整个过程可以参考下面的代码：

- 客户端同步服务端异步


```
Future future = request(server);//server立刻返回future
synchronized(future){
    while(!future.isDone()){
        future.wait();//server处理结束后会notify这个future，并修改isdone标志
    }
}
return future.get();
```

- 客户端同步服务端同步

```
Result result = request(server);
```

- 客户端异步服务端同步(这里用线程池的方式)

```
Future future = executor.submit(new Callable(){public void call(){
    result = request(server);
}});
return future;
```

- 客户端异步服务端异步

```
Future future = request(server);//server立刻返回future
return future
```

● 批量

- 攒够了一定数量。

- 到达了一定时间。

- 队列里有新的数据到来。

- 对于及时性要求高的数据，可用采用方式3来完成，比如客户端向服务端投递数据。只要队列有数据，就把队列中的所有数据刷出，否则将自己挂起，等待新数据的到来。在第一次把队列数据往外刷的过程中，又积攒了一部分数据，第二次又可以形成一个批量

```
Executor executor = Executors.newFixedThreadPool(4);
final BlockingQueue queue = new ArrayBlockingQueue();
private Runnable task = new Runnable(){//这里由于共享队列，Runnable可以复用，故做成全局的
    public void run(){
        List messages = new ArrayList(20);
        queue.drainTo(messages, 20);
        doSend(messages);//阻塞，在这个过程中会有新的消息到来，如果4个线程都占满，队列就有机会囤新的消息
    }
};
public void send(Message message){
    queue.offer(message);
    executor.submit(task)
}
```

这种方式是消息延迟和批量的一个比较好的平衡，但优先响应低延迟。延迟的最高程度由上一次发送的等待时间决定。但可能造成问题是发送过快的话批量的大小不够满足性能的极致。

- 为什么网络请求小包合并成大包会提高性能？主要原因有两个：

- 减少无谓的请求头，如果你每个请求只有几字节，而头却有几十字节，无疑效率非常低下。
- 减少回复的ack包个数。把请求合并后，ack包数量必然减少，确认和重发的成本就会降低

● push 还是 pull

上文提到的消息队列，大多是针对push模型的设计。现在市面上有很多经典的也比较成熟的pull模型的消息队列，如Kafka、MetaQ等。这跟JMS中传统的push方式有很大的区别，可谓另辟蹊径。我们简要分析下push和pull模型各自存在的利弊。

● 慢消费

慢消费无疑是push模型最大的致命伤，穿成流水线来看，如果消费者的速度比发送者的速度慢很多，势必造成消息在broker的堆积。假设这些消息都是有用的无法丢弃的，消息就要一直在broker端保存。当然这还不是最致命的，最致命的是broker给consumer推送一堆consumer无法处理的消息，consumer不是reject就是error，然后来回踢皮球。反观pull模式，consumer可以按需消费，不用担心自己处理不了的消息来骚扰自己，而broker堆积消息也会相对简单，无需记录每一个要发送消息的状态，只需要维护所有消息的队列和偏移量就可以了。所以对于建立索引等慢消费，消息量有限且到来的速度不均匀的情况，pull模式比较合适。

● 消息延迟与忙等

这是pull模式最大的短板。由于主动权在消费方，消费方无法准确地决定何时去拉取最新的消息。如果一次pull取到了还可以继续去pull，如果没有pull取到则需要等待一段时间重新pull。但等待多久就很难判定了。你可能会说，我可以有xx动态pull取时间调整算法，但问题的本质在于，有没有消息到来这件事情决定权不在消费方。也许1分钟内连续来了1000条消息，然后半个小时没有新消息产生，可能你的算法算出下次最有可能到来的时间点是31分钟之后，或者60分钟之后，结果下条消息10分钟后到了，是不是很让人沮丧？当然也不是说延迟就没有解决方案了，业界较成熟的做法是从短时间开始（不会对broker有太大负担），然后指数级增长等待。比如开始等5ms，然后10ms，然后20ms，然后40ms.....直到有消息到来，然后再回到5ms。即使这样，依然存在延迟问题：假设40ms到80ms之间的50ms消息到来，消息就延迟了30ms，而且对于半个小时来一次的消息，这些开销就是白白浪费的。在阿里的RocketMq里，有一种优化的做法-长轮询，来平衡推拉模型各自的缺点。基本思路是:消费者如果尝试拉取失败，不是直接return,而是把连接挂在那里wait,服务端如果有新的消息到来，把连接notify起来，这也是不错的思路。但海量的长连接block对系统的开销还是不容小觑的，还是要合理的评估时间间隔，给wait加一个时间上限比较好~

● 顺序消息

如果push模式的消息队列，支持分区，单分区只支持一个消费者消费，并且消费者只有确认一个消息消费后才能push送另外一个消息，还要发送者保证全局顺序唯一，听起来也能做顺序消息，但成本太高了，尤其是必须每个消息消费确认后才能发下一条消息，这对于本身堆积能力和慢消费就是瓶颈的push模式的消息队列，简直是一场灾难。反观pull模式，如果要做到全局顺序消息，就相对容易很多：

1. producer对应partition，并且单线程。

2. consumer对应partition，消费确认（或批量确认），继续消费即可。

所以对于日志push送这种最好全局有序，但允许出现小误差的场景，pull模式非常合适。如果你不想看到通篇乱套的日志~~ Anyway，需要顺序消息的场景还是比较有限的而且成本太高，请慎重考虑。

本文从为何使用消息队列开始讲起，然后主要介绍了如何从零开始设计一个消息队列，包括RPC、事务、最终一致性、广播、消息确认等关键问题。并对消息队列的push、pull模型做了简要分析，最后从批量和异步角度，分析了消息队列性能优化的思路。下篇会着重介绍一些高级话题，如存储系统的设计、流控和错峰的设计、公平调度等。希望通过这些，让大家对消息队列有个提纲挈领的整体认识，并给自主开发消息队列提供思路。另外，本文主要是源自自己在开发消息队列中的思考和读源码时的体会，比较不“官方”，也难免会存在一些漏洞，欢迎大家多多交流。

后续我们还会推出消息队列设计高级篇，内容涵盖以下方面：> * pull模型消息系统设计理念 > * 存储子系统设计 > * 流量控制 > * 公平调度