https://yq.aliyun.com/articles/2892

metaq是什么?

是一个基于“发布-订阅”的队列模型消息中间件，服务端使用JAVA编写，客户端支持JAVA、C++。

对外已开源，名字叫RocketMQ。

metaq的特性

消费模型

metaq采用发布-订阅模型，发布者发布消息到metaq，订阅者向metaq订阅消息。

消息的消费方式是pull方式，由消费者主动从metaq服务器拉取数据，解析成消息并消费。

消息持久性

metaq 接收到消息之后，会先把消息持久化到本地。

常用的持久化方式：

- 持久化到DB
- 持久化到KV存储，如levelDB，伯克利DB
- 持久化到文件

metaq使用的是 **持久化到文件** ，并充分利用Linux文件系统内存cache来提高性能。

注： *持久化部分的性能会直接影响消息中间件的性能。*

消息堆积能力：metaq每台服务器提供大约亿级的消息堆积能力（多个业务方共用），超过堆积阈值，订阅消息吞吐量会下降。

消息过滤

对于应用比较多，访问量比较大的情况，消息量也就随之增大，一方面服务端给每个客户端发送消息时，总不能把全站的消息都发送过去，这样大量的无用消息在网络上传输是一种资源浪费。另一方面，客户端也不需要接收所有的消息，而只需要接收自己需要的消息。这时，消息中间件就需

要一个消息过滤的功能。

metaq支持两种过滤方式：服务器端过滤，客户端过滤。

- 服务器端过滤： **优点** 是减少网络上无用消息的传输， **缺点** 是增加服务端负担，实现复杂
- 客户端过滤： **优点** 是可以完全根据自己的需要定制哪些需要哪些不要， **缺点** 是很多无用的消息要传输到客户端。如果客户端对应的应用不足以支撑这么多消息，就会造成应用的所有计算资源全部都在处理这些消息，甚至拖垮整个应用。

消息实时性

metaq客户端通过长轮询的方式连接服务端，可以保证消息非常实时，实时性不低于push

每个消息至少投递一次

Consumer先pull消息到本地，消费完之后，才会向服务器返回。

为了追求性能，metaq并不保证消息不重复发送，但是正常情况下很少出现。只有网络异常，consumer启动、停止等异常情况下才会出现重复。

本质原因是网络调用的不确定性，即会出现既不成功也不失败的第三种状态。

保证消息局部有序

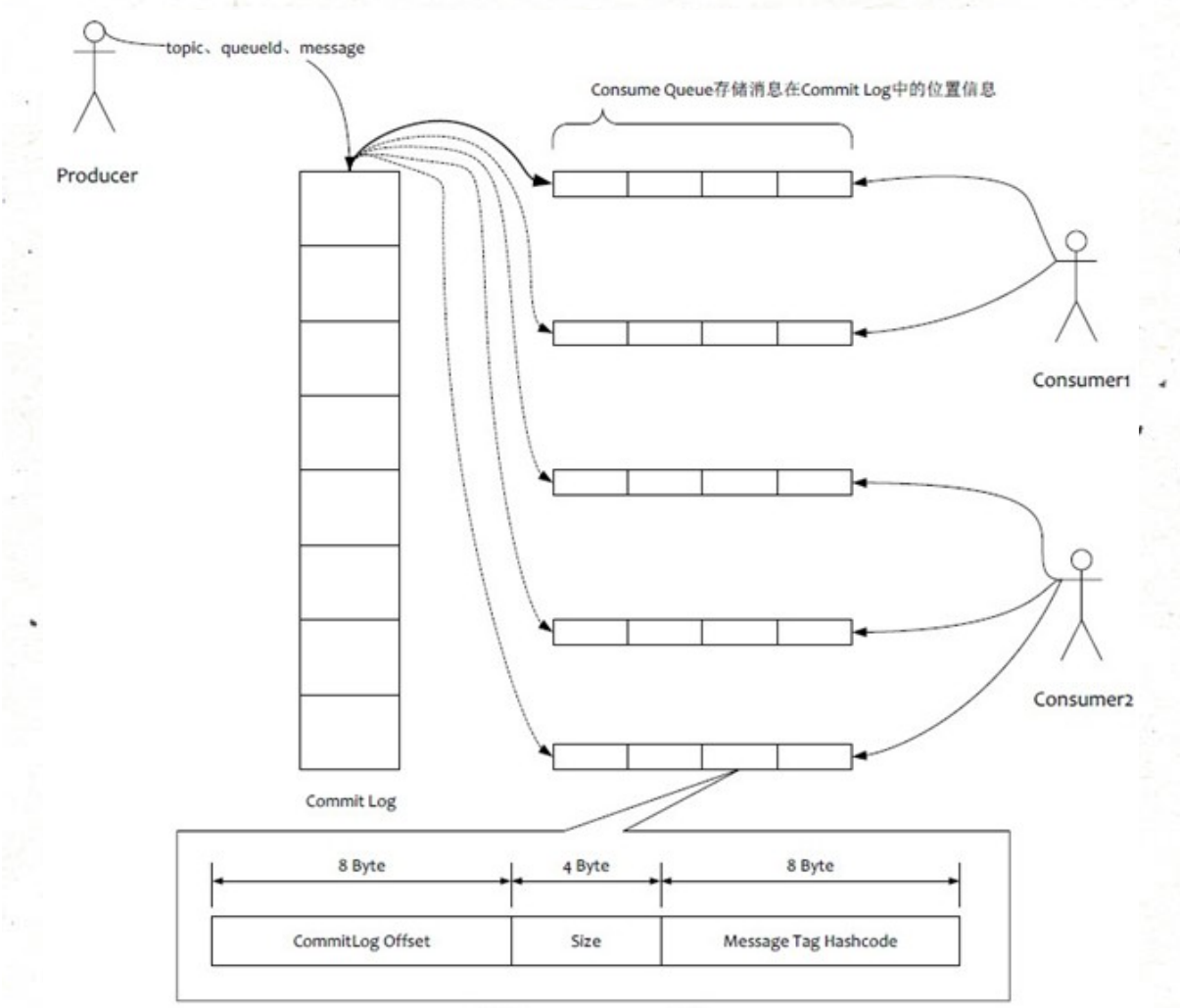
通过队列的特性，保证消息的顺序。

发送端，将需要保证顺序的消息发送到同一个队列中。消费端，从队列里取消息，顺序消费。

不同的几组消息，可以发送到不同的队列中，提高并行性。

metaq的存储结构

metaq的逻辑存储结构是一种物理队列+逻辑队列的结构。



物理队列只有一个，采用固定大小的文件顺序存储消息。逻辑队列有多个，每个逻辑队列有多个分区，每个分区有多个索引。

a.消息顺序写入物理文件里面，每个文件达到一定的大小，新建一个文件继续顺序写数据（消息的写入是串行的，避免了磁盘竞争）。

b.消息的索引则顺序的写入逻辑文件中，并不存放真正的消息，只是存放指向消息的索引。

metaq对于客户端展现的是逻辑队列就是消费队列，consumer从消费队列里顺序取消息进行消费。

这种设计是把物理和逻辑分离，消费队列更加轻量化。所以metaq可以支撑更多的消费队列数，提升消息的吞吐量，并且有一定的消息堆积能力。

缺点：

写虽然是顺序写，但是读却是随机读的

解决办法： 尽可能让读命中pageCache，减少磁盘IO次数 *(参考下文所述：Linux的文件Cache管理)*

metaq的所有消息都是持久化的，先写入系统PAGECACHE（页高速缓存），然后刷盘，可以保证内存与磁盘都有一份数据，访问时，直接从内存读取。

刷盘策略分为异步和同步两种。

Linux的文件Cache管理

在 Linux 操作系统中，为了加快文件的读写，当应用程序需要读取文件中的数据时，操作系统先分配一些内存，将数据从存储设备读入到这些内存中，然后再将数据分发给应用程序；当需要往文件中写数据时，操作系统先分配内存接收用户数据，然后再将数据从内存写到磁盘上。

文件 Cache 管理就是对这些由操作系统分配，并用来存储文件数据的内存的管理。

Cache 管理的优劣通过两个指标衡量：

- Cache 命中率：Cache 命中时数据可以直接从内存中获取，不再需要访问低速外设，因而可以显著提高性能；
- 有效 Cache 的比率：有效 Cache 是指真正会被访问到的 Cache 项。如果有效 Cache 的比率偏低，则相当部分磁盘带宽会被浪费到读取无用 Cache 上，而且无用 Cache 会间接导致系统内存紧张，最后可能会严重影响性能。 在 Linux 的实现中，文件 Cache 分为两个层面，一是 Page Cache，另一个 Buffer Cache。 每一个 Page Cache 包含若干 Buffer Cache。

通过内存映射的方式读写文件

metaq在文件读写操作上做了一定的优化，使用内存映射的方式完成读写，替代了传统的IO操作，从而大大的减少了文件读写系统调用的次数，提升了IO的性能。

传统的文件访问：

- 系统调用打开文件，获取文件描述符
- 使用read write 系统调用进行IO
- 系统调用关闭文件

这种方式是非常低效的, 每一次I/O操作都需要一次系统调用。 另外, 如果若干个进程访问同一个文件, 每个进程都要在自己的地址空间维护一个副本, 浪费了内存空间

内存映射的方式：

- 打开文件，得到文件描述符。
- 获取文件大小
- 把文件映射成虚拟内存（mmap）
- 通过对内存的读写来实现对文件的读写（memset或memcpy）
- 卸载映射
- 关闭文件

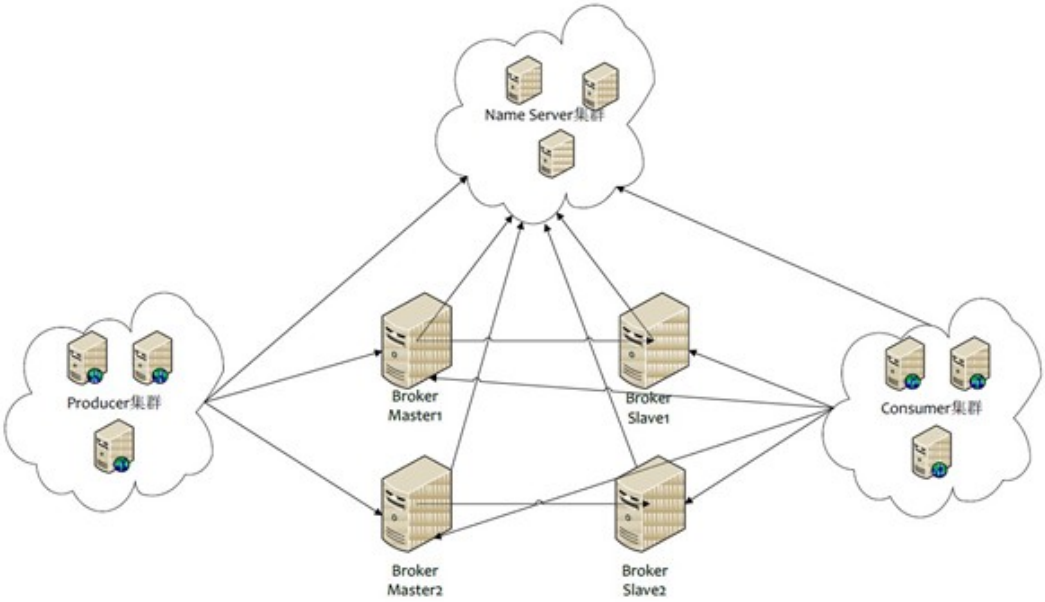
首先建立好虚拟内存和磁盘文件之间的映射（mmap系统调用），当进程访问页面时产生一个缺页中断，内核将页面读入内存(也就是说把磁盘上的文件拷贝到内存中)，并且更新页表指向该页面。

所有进程共享同一物理内存，物理内存中可以只存储一份数据，不同的进程只需要把自己的虚拟内存映射过去就可以了，这种方式非常方便于同一副本的共享，节省内存。

经过内存映射之后，文件内的数据就可以用内存读/写指令来访问，而不是用Read和Write这样的I/O系统函数，从而提高了文件存取速度。

metaq架构&消息的收发

metaq的整体架构如下图所示，主要包括Broker集群（metaq的服务端），client集群（发布者集群和订阅者集群），nameServer集群。



Broker分为master和slave。每个Broker与nameserver集群中的所有节点建立长连接，定时注册topic信息到所有的nameServer。

Producer与nameServer集群中的一个节点（随机）建立长连接，定期从nameServer 取topic路由信息，并向提供topic服务的master broker建立长连接，且定时向master发送心跳。Producer发布消息是发布到master，在由master同步到所有broker。

Consumer与nameServer集群中的一个节点建立长连接，定期从nameServer取topic的路由信息，并向提供topic服务的master、slave broker建立长连接，并定时向master、slave发送心跳。Comsumer既可以从slave订阅消息，也可以从master订阅消息。

保证消息的可靠性

一个消息从发送端应用，到消费端应用，中间有三个过程需要保证消息的可靠性。

1.发送端发消息

消息生产者发送消息后返回SendResult，如果isSuccess返回为true,则表示消息已经确认发送到服务器并被服务器接收存储。整个发送过程是一个同步的过程。保证消息送达服务器并返回结果。

只有当消息中间件及时明确的返回成功，才能确认消息可靠到达消息中间件。

2.消息中间件把消息存储起来

metaq服务器收到消息后首先把消息存放在磁盘文件中，确保持久存储，写入成功之后返回应答给发布者。因此，可以确认每条发送结果为成功的消息服务器都是写入磁盘的。

内存中内容属于非持久数据，会在断电之后丢失。

3.消费端消费消息

消费者是一条接着一条地顺序消费消息，只有在成功消费一条消息后才会接着消费下一条。

如果在消费某条消息失败（如异常），则会尝试重试消费这条消 息（默认最大5次），超过最大次数后仍然无法消费，则将消息存储在消费者的本地磁盘，由后台线程继续做重试。而主线程继续往后走，消费后续的消息。。由此来保证消息的可靠消费。

metaq消息底层通信组件

metaq消息的传递，通信，是使用的netty，并在netty之上作了简单的协议封装。

网络协议如下:

数据部分采用json序列化。

metaq的主要应用场景

1.消息推送

许多系统通过metaq进行消息推送

2.数据库同步

精卫是通过metaq发消息感知数据库binlog的变化，并进行数据库复制的。

精卫是阿里的一个数据库同步中间件。

精卫首先解析mysql的binlog，然后以消息的形式发往metaq，下游应用（比如终搜，TC，IC等）来消费Mysql数据库操作的变更事件完成数据库同步。

整个过程，metaq通过提供严格的顺序消息，事务消费方式保证了数据的

可靠，高效。

3.实时消息

IM对消息实时性要求极高，metaq目前在来往得到了广泛使用，包括注册通知、私信、扎堆分享，语音文字消息等功能在使用metaq。

metaq使用长轮询拉模式，可保证消息同push方式一样实时，通常在几个毫秒。

metaq与notify的对比

notify	metaq
消息不保证100%有序	消息有序
push模式（服务端主动推）	pull模式（客户端主动拉）
支持分布式事务	不支持
基于jms标准	基于消息队列
可以选择持久化or非持久化	所有消息都是持久化的
不支持消息回溯，只能重发	支持消息回溯
不支持集群内广播消息	支持（集群内每台机器都能收到消息）
适合比较复杂的业务模型	轻量级，高性能
接入方式更灵活	接入方式不如Notify灵活

Push or Pull:

- Push模式：很难掌握消息推送的时机和速率，因为consumer的消费

速率不同。

- Pull模式：consumer可以根据自己的状况选择拉取消息的时机和速率，缺点在于如果服务端没有可供消费的消息，将导致consumer不断轮询，浪费资源。