RocketMQ

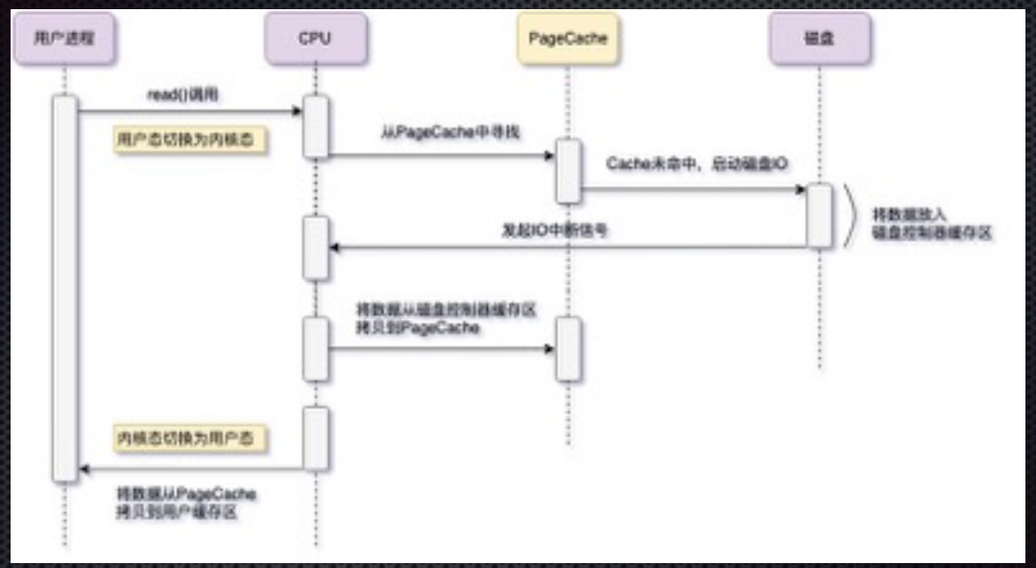
yongping.ren@ygomi.com

- PageCache
- MMAP
- ConsumeQueue
- consumer负载均衡
- IndexFile

PageCache

- 页缓存（PageCache)是OS对文件的缓存，用于加速对文件的读写。一般来说，程序对文件进行顺序读写的速度几乎接近于内存的读写速度，主要原因就是由于OS使用PageCache机制对读写访问操作进行了性能优化。

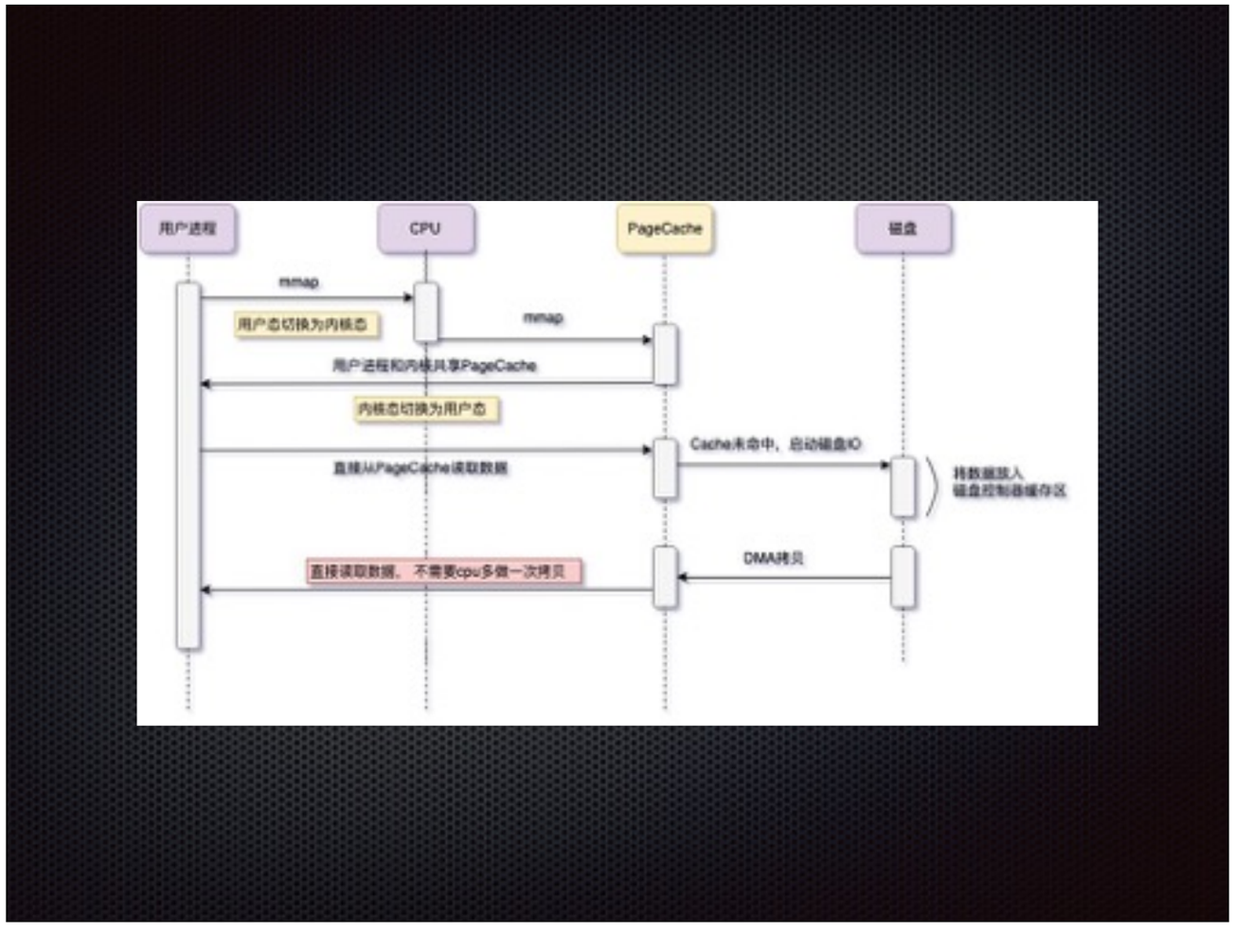
- 对于数据的写入：OS会先写入至Cache内，随后通过异步的方式由pdflush内核线程将Cache内的数据刷盘至物理磁盘上。
- 对于数据的读取：如果一次读取文件时出现未命中PageCache的情况，OS从物理磁盘上访问读取文件的同时，会顺序对其他相邻块的数据文件进行预读取。



PageCache工作流程

Mmap

- 将磁盘上的物理文件直接映射到用户态的内存地址中（这种Mmap的方式减少了传统IO将磁盘文件数据在操作系统内核地址空间的缓冲区和用户应用程序地址空间的缓冲区之间来回进行拷贝的性能开销），将对文件的操作转化为直接对内存地址进行操作，从而极大地提高了文件的读写效率（正因为需要使用内存映射机制，故RocketMQ的文件存储都使用定长结构来存储，方便一次将整个文件映射至内存）


用户进程
CPU
PageCache
磁盘
mmap
用户态切换为内核态
mmap
用户进程和内核共享PageCache
内核态切换为用户态
Cache未命中，启动磁盘IO
直接从PageCache读取数据
将数据放入
磁盘控制器缓存区
DMA拷贝
直接读取数据，不需要cpu多做一次拷贝

mmap工作流程

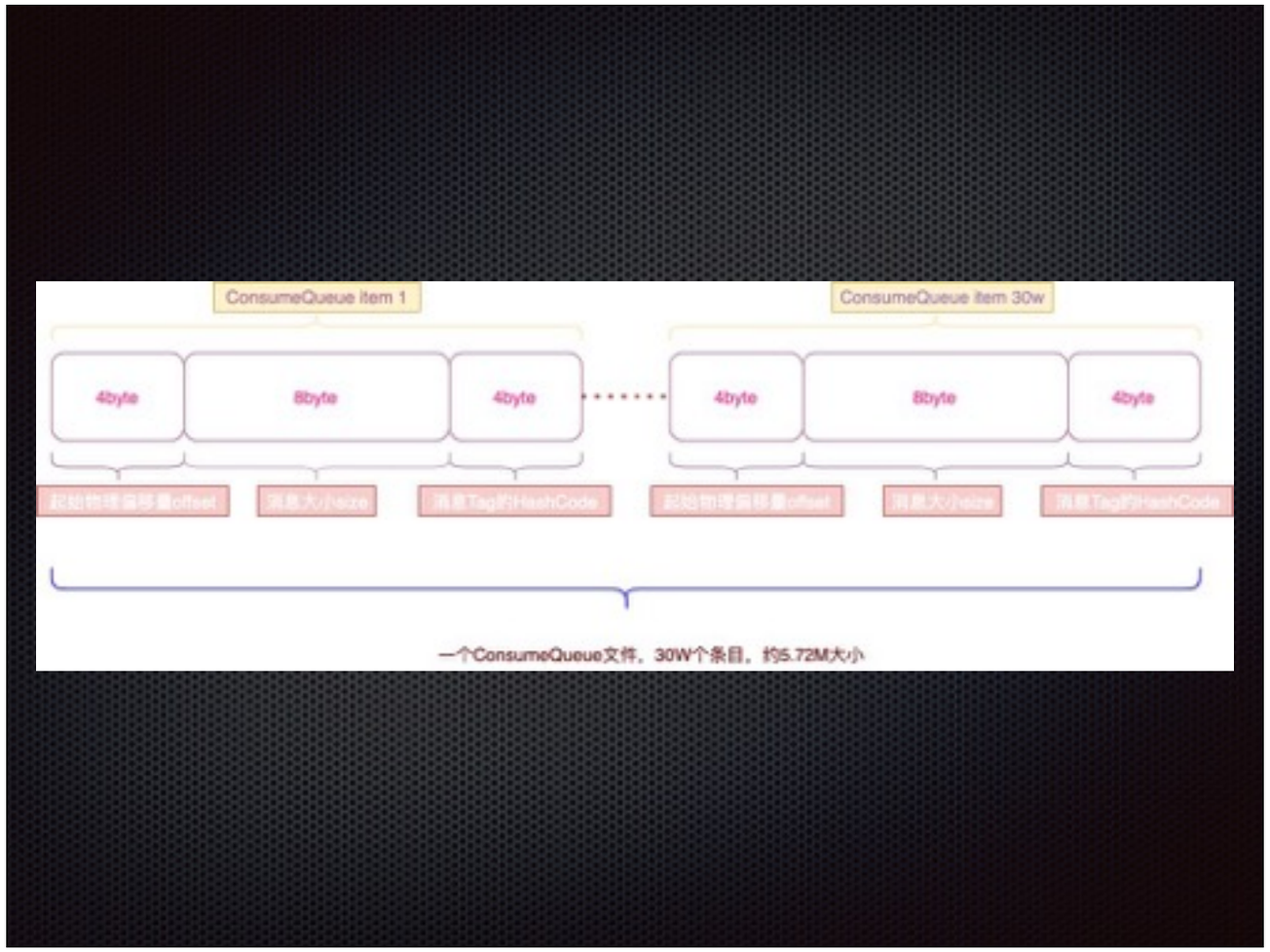
总结

- page cache的预读取及异步刷盘机制增强读写性能
- Mmap的方式减少内核地址空间和用户地址空间来回进行拷贝的性能开销

ConsumeQueue

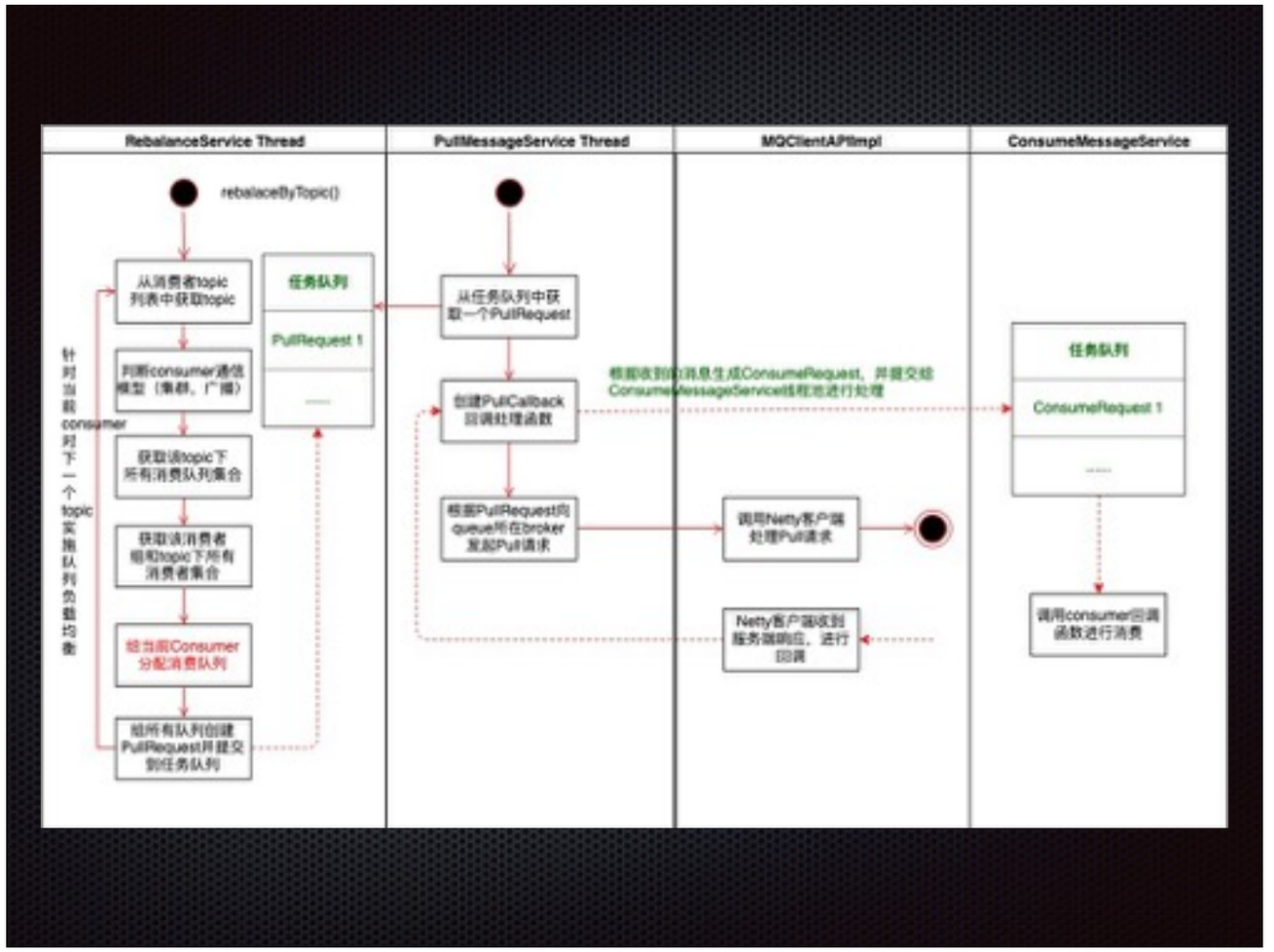
- 引入的目的主要是提高消息消费的性能，ConsumeQueue（逻辑消费队列）作为消费消息的索引，保存了指定Topic下的队列消息在CommitLog中的起始物理偏移量offset，消息大小size和消息Tag的HashCode值。consumequeue文件可以看成是基于topic的commitlog索引文件。

引入的目的主要是提高消息消费的性能（顺序读取，利用PageCache的预读取机制实现的）


ConsumeQueue item 1
ConsumeQueue item 30w
4byte
8byte
4byte
4byte
8byte
4byte
一个ConsumeQueue文件，30W个条目，约5.72M大小

Consumer负载均衡

- RocketMQ中的负载均衡在Client端完成,具体来说，主要可以分为Producer端发送消息时候的负载均衡和Consumer端订阅消息的负载均衡(Consumer端在消费消息时，需要知道从Broker端的哪一个消息队列—队列中去获取消息)。

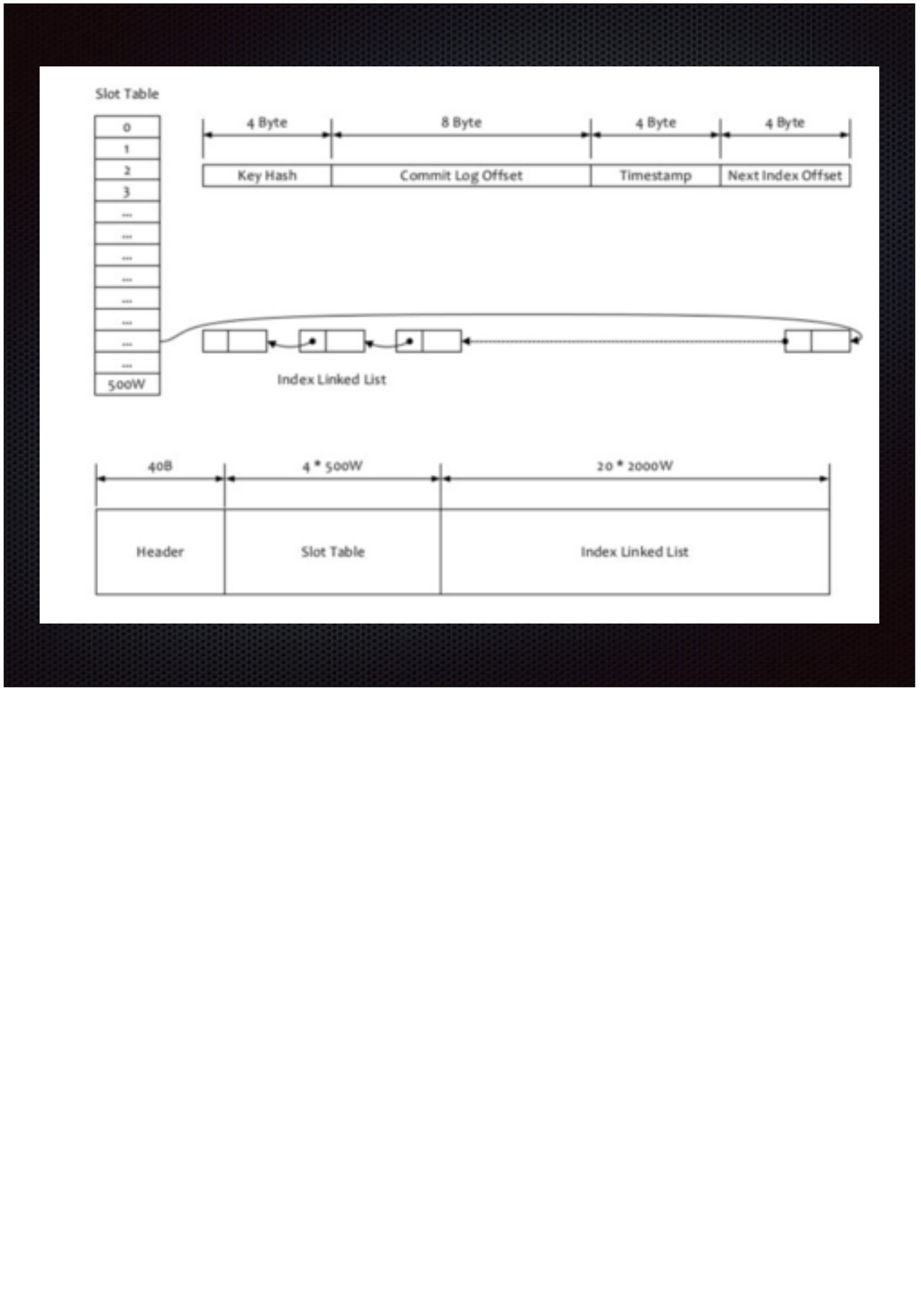

RebalanceService Thread
PullMessageService Thread
MQClientAPIImpl
ConsumeMessageService
rebalaceByTopic()
从消费者topic列表中获取topic
判断consumer通信模型（集群，广播）
获取该topic下所有消费队列集合
获取该消费者组和topic下所有消费者集合
给当前Consumer分配消费队列
给所有队列创建PullRequest并提交到任务队列
针对当前consumer对下一个topic实施队列负载均衡
任务队列
PullRequest 1
.....
从任务队列中获取一个PullRequest
创建PullCallback回调处理函数
根据PullRequest向queue所在broker发起Pull请求
调用Netty客户端处理Pull请求
Netty客户端收到服务端响应，进行回调
根据收到的消息生成ConsumeRequest，并提交给ConsumeMessageService线程池进行处理
任务队列
ConsumeRequest 1
.....
调用consumer回调函数进行消费

总结

- RocketMQ默认采取平均分页算法在客户端完成对消费者的负载均衡。

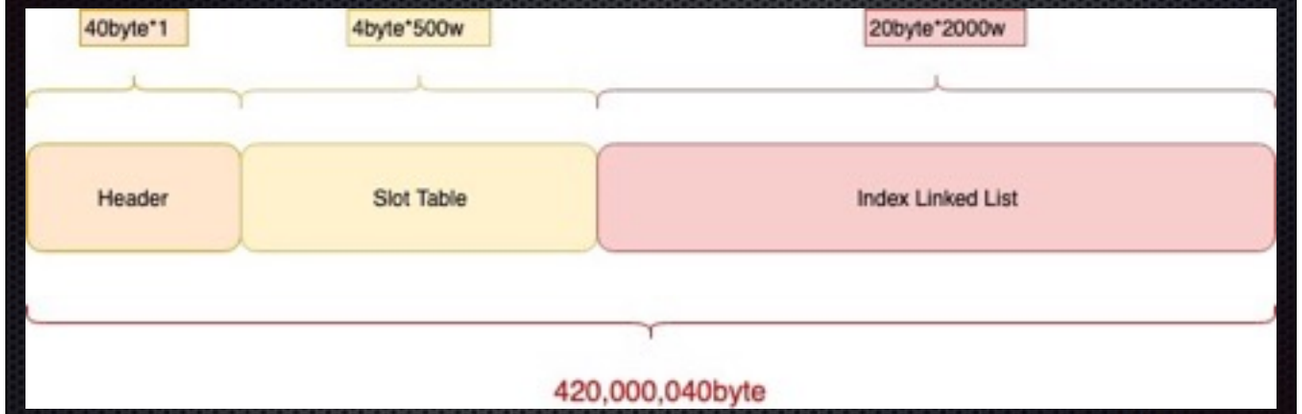
IndexFile

- IndexFile索引文件为用户提供通过“按照Message Key查询消息”的消息索引查询服务，RocketMQ的索引文件逻辑结构，类似JDK中HashMap的实现。

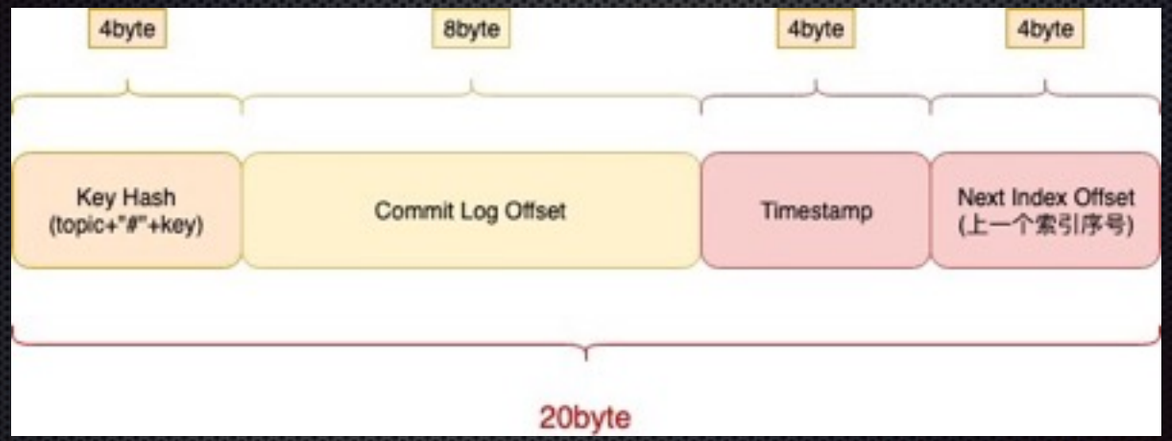

Slot Table
0
1
2
3
...
...
...
...
...
...
...
...
500W
4 Byte
8 Byte
4 Byte
4 Byte
Key Hash
Commit Log Offset
Timestamp
Next Index Offset
Index Linked List
40B
4 * 500W
20 * 2000W
Header
Slot Table
Index Linked List

官方图解

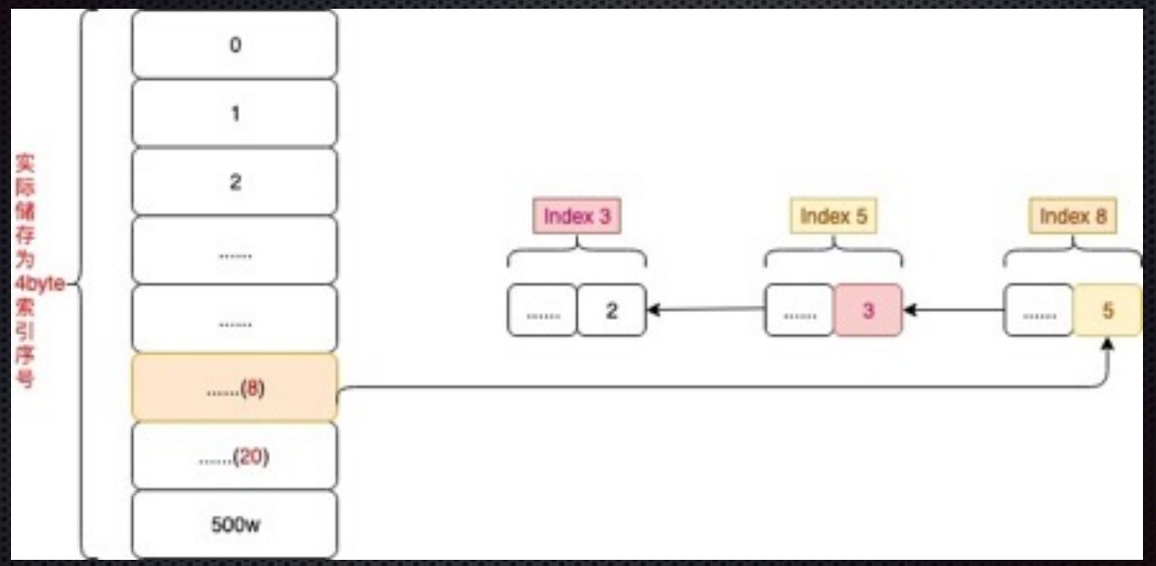
IndexFile结构



索引链表节点


4byte
8byte
4byte
4byte
Key Hash
(topic+"#"+key)
Commit Log Offset
Timestamp
Next Index Offset
(上一个索引序号)
20byte

索引槽



总结

- IndexFile使用类似于Java HashMap的实现原理实现对消息根据关键字的存储和查询

Thank you!