RocketMQ STORE Q&A

v3.0.0

2013/10/7

文档变更历史

序号	主要更改内容	更改人	更改时间
1	建立初始版本	誓嘉，韩彰 vintage.wang@gmail.com	2012/11/20
2	3.0 版本补充文档	誓嘉 vintage.wang@gmail.com	2013/8/18
3			
4			
5			
6			
7			

目录

1 写入 Commit Log 成功，但是写入 Consume Queue 失败怎么办？

- JVM CRASH 情况下，消息位置信息未写入 Consume Queue，如何处理？

  JVM 重启后，从 Commit Log 恢复 Consume Queue，非全量恢复，只恢复当前可能丢失的数据

- 写入 Consume Queue 发生 IO 错误如何处理？

  一旦发生 IO 错误，则认为可能是 IO 设备故障，停止对外写服务，但是数据仍然可读。

- JVM 发生 outofmemory

  这种情况，可理解为 jvm 不能对外服务，Consume Queue 与 Commit Log 可能不一致。必须重启才能保证消息一致。

2 写完 Commit Log 后，消息位置信息是同步写入 Consume Queue 还是异步写入？

消息一旦写入 Commit Log，则返回消息对应的 CommitLog offset，size，等信息，将这类消息位置信息传递至另一个独立的 Dispatch 线程，然后主流程返回，并向发送方返回成功应答。

3 异步写消息位置信息是单线程还是多线程？

单线程，可保证消息顺序。

4 Consumer 拉消息是单个方式还是批量方式拉？

是批量方式拉消息，服务器可以配置一次最多拉多少条，最多多少字节，客户端也可以配置。

以最小的为主。

5 存储层大量使用了 mmap 内存文件映射，是否会超出系统句柄限制？

不会。

系统默认值

vm.max_map_count = 65536

可以通过以下方式修改

sudo sysctl vm.max_map_count=655360

同时需要关注以下参数

[shijia.wxr@dev170021 ~]$ ulimit -a

open files (-n) 131072

6 异步刷盘方式，是否会导致内存爆掉？

不会。

消息写入 pagecache 后，直接通知刷盘线程开始刷盘，并返回，也就是说刷盘线程处于实时刷盘状态。但是也不排除前端发消息压力大、而且后端消费消息堆积程度严重，造成消息不能及时刷盘，在内存堆积过多的情况。这种情况下，当内存未刷盘数据达到一定程度，就开始阻塞前端写 pagecache 操作。（实际是每次都尝试刷盘 32 个 page 才返回）。具体阀值参见以下参数。

vm.dirty_bytes=20000000000

7 Consumer 拉消息是否使用了 sendfile？

Server 在向 Consumer 返回消息时，没有直接使用 sendfile（transferTo）接口，但是使用了类似机制，直接将虚拟内存地址传给 socket，无论实际数据是在物理内存还是在文件，都由操作系统进行管理，也就是说消

息数据不会重新 load 到 java 堆。

另外：将 pagecache 直接传输给 socket，在操作系统层面会做以下优化

- 因为 pagecache 内存本身是内核与应用共享的内存，所以不需要用户态向内核态内存拷贝。
- Socket 在发现是 pagecache 时，会将 pagecache 直接传输，不需要将数据拷贝到 socket 缓冲区。（TCP 协议栈优化）

8 所有文件都映射至内存，所占用的物理内存比例如何控制？

可通过以下参数来控制 server 占用的物理内存比例

sudo sysctl vm.min_free_kbytes=7000000

9 PAGECACHE 使用过多，是否会影响 swap？

Pagecache 与 swap 是两个独立的机制，即使映射 1T 的 pagecache，也不会有过多的 swap，且可以通过以下命令来关闭系统的 swap 机制。

sudo swapoff –a

也可以设定以下参数，令系统尽可能不要 swap

sudo sysctl vm.swappiness=0

10 JVM CRASH 后，写入 PAGECACHE 的未刷盘数据是否会丢失？

不会。

为了避免 os 本身的刷盘机制与应用自己的刷盘机制冲突，设置了以下参数来抑制 os 刷盘。

sudo sysctl vm.dirty_writeback_centisecs=360000

但是同时会导致 JVM CRASH 后，系统的 pagecache 不能及时刷盘，此时可以通过以下命令来刷盘

sync

11 PAGECACHE 映射总大小是否有限制？

32 位 linux 存在地址空间 4G 的限制，所以 RocketMQ 不适合在 32 位机器上运行。

Win32 默认限制为 2G，更不适合。

关于在 64 位机器上，pagecache 是否有限制？

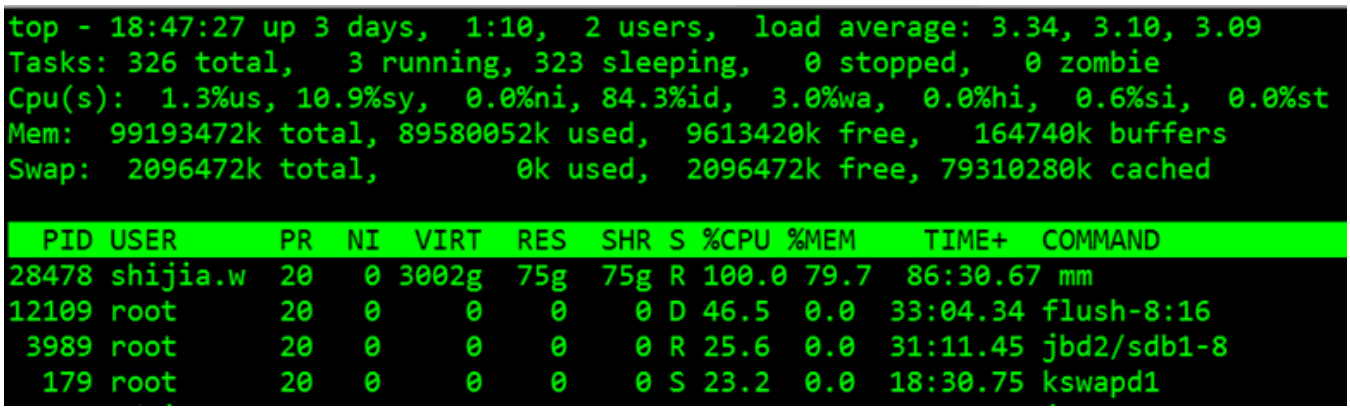
理论上，64 位机器地址空间可认为无限大（但是实际由于 cpu 地址空间限制，可能会小于 2 的 64 次方）。

以下为线上 96G 64 位 linux，3.2T 磁盘空间机器运行截图

注：是用 c 写的一个简单测试程序，不断尝试调用 mmap 接口，并且向里面写数据，只有写数据才会实际分配内存地址。

代码详见以下网址：

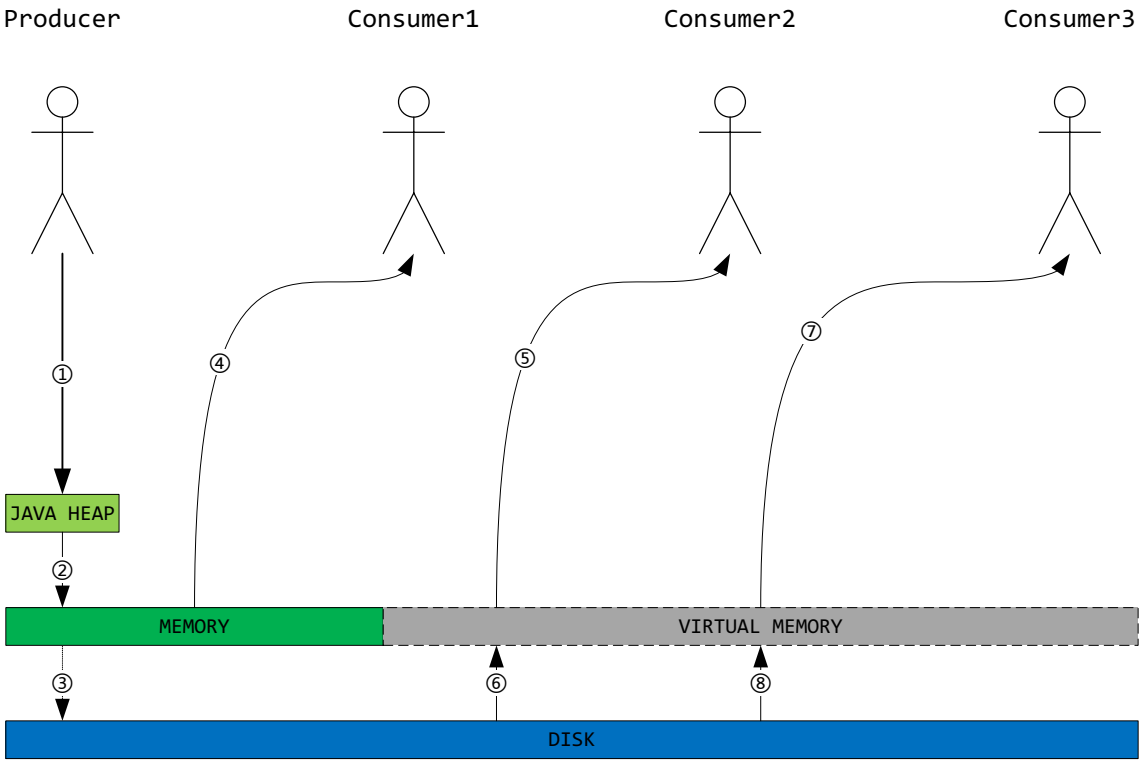
https://gist.github.com/3735144



12 是否可以认为 Linux64 位可以无限制映射 PAGECACHE？

Pagecache 是由内核维护的，映射的越多，内核数据结构占用的物理内存越大，所以要映射更多的 pagecache，对机器的物理内存大小要求更高。

13 消息如何在 java 堆，物理内存，虚拟内存，磁盘之间流动？



图表 1 消息数据流图（机器视图）

(1). Producer 发送消息，消息从 socket 进入 java 堆。

(2). Producer 发送消息，消息从 java 堆转入 PAGACACHE，物理内存。

(3). Producer 发送消息，由异步线程刷盘，消息从 PAGECACHE 刷入磁盘。

(4). Consumer 拉消息（正常消费）消息直接从 PAGECACHE（数据在物理内存）转入 socket，到达 consumer，不经过 java 堆。

这种消费场景最多，线上 96G 物理内存，按照 1K 消息算，可以在物理内存缓存 1 亿条消息。

(5). Consumer 拉消息（异常消费），消息直接从 PAGECACHE（数据在虚拟内存）转入 socket。

(6). Consumer 拉消息（异常消费），由于 Socket 访问了虚拟内存，产生缺页中断，此时会产生磁盘 IO，从磁盘 Load 消息到 PAGECACHE，然后直接从 socket 发出去。

(7). 同 5 一致。

(8). 同 6 一致。

14 Commit Log 与 Consume Queue 使用 long 类型标识 offset，是否会溢出？

参见下表

消息大小	每天单台 SERVER 消息量（单位亿）	单台 RocketMQ SERVER 可持续运行时间（单位年）
256	10	98709.03293
512	10	49354.51646
1024	10	24677.25823
2048	10	12338.62912
4096	10	6169.314558
256	100	9870.903293
512	100	4935.451646
1024	100	2467.725823
2048	100	1233.862912
4096	100	616.9314558
256	1000	987.0903293
512	1000	493.5451646
1024	1000	246.7725823
2048	1000	123.3862912
4096	1000	61.69314558

由此可知，单台 RocketMQ server，按照 4K 消息，一天接收 100 亿条消息，可以连续运行 616 年

15 RocketMQ Server 关闭时，数据安全性如何保证？

A. 正常关闭，一般通过人工调用 kill -15 pid 形式关闭，Server 内部会捕获 SIGTERM 信号，并进行处理，将内存数据全部刷盘。

B. 异步刷盘情况下，异常关闭（重启时，必须要进行纠错）

a) Kill -9 形式关闭，由于程序无法捕获-9 信号，会被非法关闭。

此时向 Commit Log 写消息可能会只写入半个消息，Consume Queue 同样也存在这种情况，都是最后一个消息无法保证正常写入。

但是之前写入完整的消息虽然未刷盘，也可以保证不丢失，数据只要进入 PAGECACHE，即使程序 CRASH，仍然在内存中

可以通过 sync 命令刷盘（系统内置命令）

b) OS CRASH 或机器掉电

此种情况，只要未刷盘的数据将全部丢失。

根据性能压测结果，实际在内存未刷盘数据大概在几十 K 的样子。也就是说最糟糕的情况会有几十 K 的消息丢失。

C. 同步刷盘情况下，异常关闭（重启时，必须要进行纠错）

B 中涉及的两种异常情况，都不会丢消息，但是可能存在如下情况

当消息写入到 pagecache，刷盘刷到一半时，此时还未向 Producer 返回成功，但是机器掉电或者 OS CRASH,这个半个消息就是脏数据，重启时需要纠错。另外 Producer 也会收到超时异常，由用户决定是否要重试。

综上，同步刷盘情况下，异常关闭不会丢消息。

16 RocketMQ Server 重启时，如何 Load 数据？

A. 上次正常退出后重启

正常退出指的是，所有内存数据都已经正常刷盘，Commit Log 与 Consume Queue 对应关系一致，恢复时各自独立恢复到内存即可。

B. 上次异常退出后重启

异常退出指的是 ,Commit Log 与 Consume Queue 可能数据不一致 ,有可能 Commit Log 比 Consume Queue 数据多，也有可能 Consume Queue 比 Commit Log 数据多，这里一 Commit Log 数据为主，从 Commit Log 上次刷盘位置开始扫描 Commit Log，将消息重新派发至 Consume Queue。

如何找到上次刷盘位置？

“checkpoint”此文件会记录刷盘的时间戳，恢复时，根据时间戳来扫描 Commit Log，就可以找到从哪里开始恢复。

如果此文件丢失，则会对 Commit Log 进行全盘扫描恢复，这种情况会耗时较长。

17 RocketMQ 如何区别是正常退出还是异常退出？

RocketMQ 启动时，都会在指定目录创建一个文件“abort”，如果正常退出，则将文件删除，如果异常退出，则没有机会删除文件，所以在 RocketMQ 重启时，只要发现这个文件存在就认为上次是异常退出，需要校验数据，如果文件不存在，则认为上次是正常退出，数据都 OK。

18 RocketMQ 有哪些自我保护措施？

- 磁盘空间使用超过 90%阀值时，Server 自动停止对外写服务，也就是发送方发消息会被拒绝。Consumer 仍然可以拉消息。
- 消息向 Consume Queue 写入失败时，尝试重试 3 次，如果仍然失败，则认为 IO 设备发生重大错误，停止对外写服务。Consumer 仍然可以拉消息。

19 RocketMQ 是否需要流控？

- 对于发送消息，接收消息不需要流控

  因为性能测试中，千兆网卡上下行同时压满（流量都在 100M 以上），系统指标仍然正常。但是同时需要监控磁盘空间剩余量，因为在高 TPS 场景下，磁盘很快就会被写满。

- Server 内部将消息消息位置信息派发至各个 Consume Queue 需要流控

  在 1 万队列以下一般不需要流控，但是一旦超过 1 万个队列，则对队列的写性能会下降，此时前端请求过来，消息位置信息会在 java 堆中堆积，默认阀值是 40 万，超过则开始流控，对前端请求做 1 毫秒 sleep。

20 RocketMQ 为什么可以支持海量（1 万以上）队列？

对队列有以下两点优化

- 队列概念轻量化，队列中不真正存储消息，只存储 20 字节的消息消息位置信息
- 刷盘方式由之前并行刷盘改为串行刷盘，避免了磁盘竞争。

21 Java 中使用 MapedByteBuffer 可能会使 JVM CRASH，RocketMQ 如何避免？

使用 MapedByteBuffer 导致 JVM CRASH 的原因是什么？

由于 MapedByteBuffer 可以通过特殊方法人为释放掉，实际调用了 unmap 方法。此时之前映射到 JVM 的地址空间就非法，如果此后仍然对 MapedByteBuffer 进行读写，系统就会向 JVM 发送 SIGBUS 信号来通知进程此种操作非法。（这种一般是由于程序员没有处理好并发问题导致）

RocketMQ 如何避免？

采用引用计数方法，参考 C++智能指针实现方式。只要引用计数不为 0，MapedByteBuffer 对象就不会释放。

为什么不使用读写锁来避免？

- 采用引用计数使用的是原子变量，并发下要比读写锁性能更好
- 采用读写锁，每次对数据读写都要加锁，代码较冗余（个人看法）。

这种方式存在什么弊端？

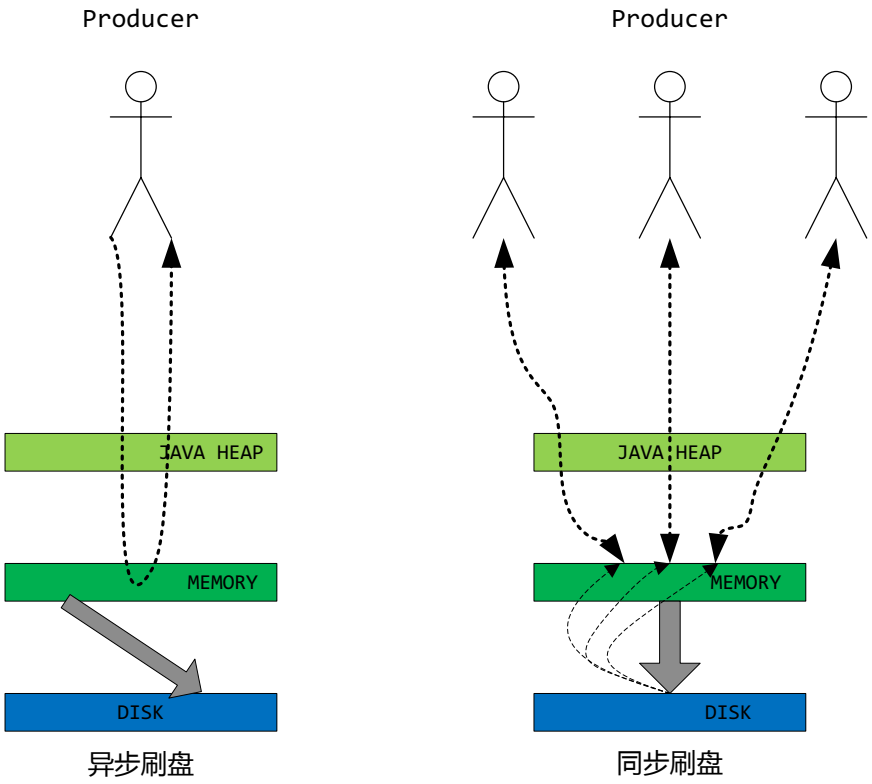
如果不能正确操作引用计数，可能会导致文件无法删除，所以 RocketMQ 增加了一个补救措施，就是一旦关闭文件服务后，如果超过 2 分钟，引用计数还没有变为 0，则强制释放。

22 订阅关系丢失，是否也会丢失消息？

不会。

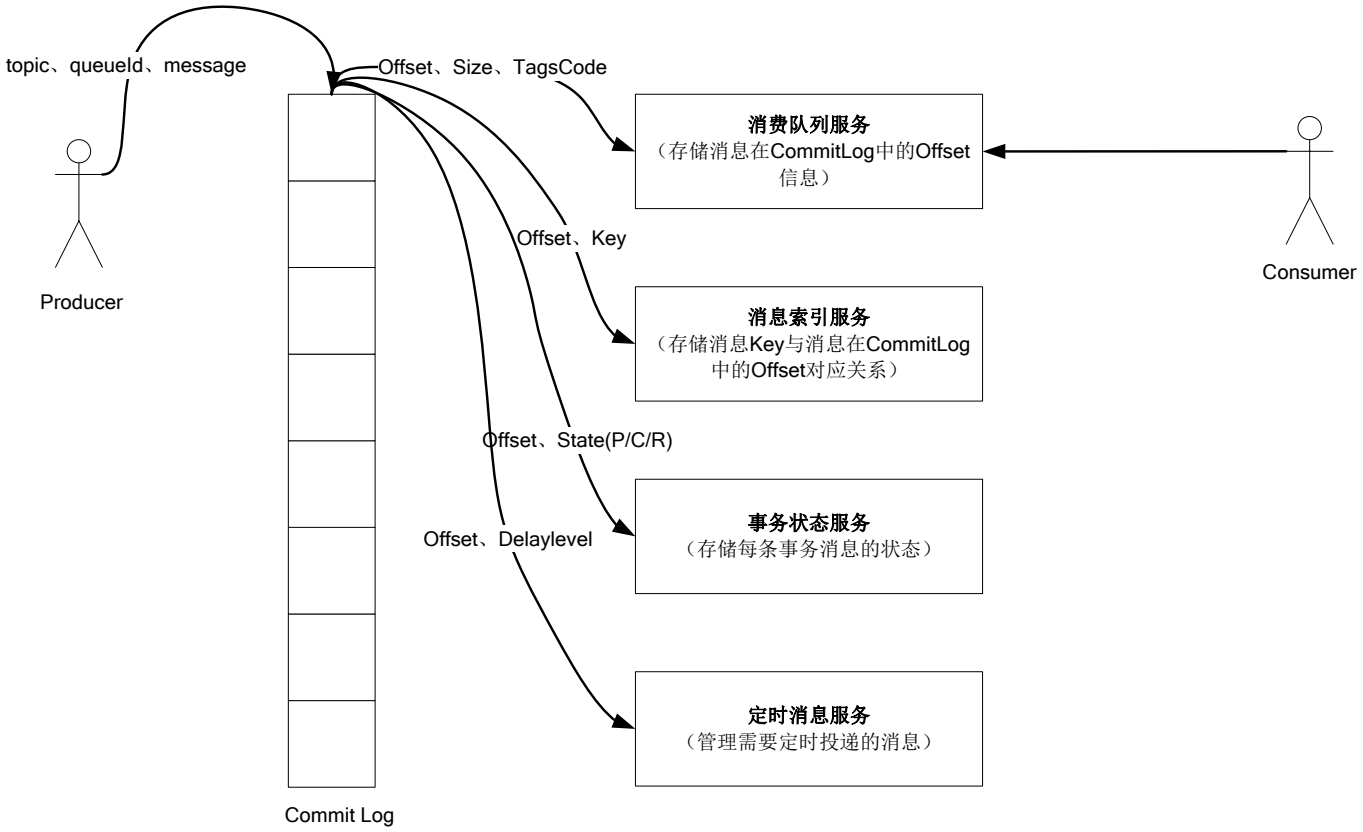
不管是否有人订阅消息，消息都在那里。

23 存储层的刷盘策略是什么？



图表 2 刷盘策略

24存储层视图？



图表 3 存储层视图