|  |
| --- |
|  |
| **RocketMQ STORE Q&A** |
|  |
|  |

**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？ 1](#_Toc364609619)

[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？ 1](#_Toc364609620)

[3 异步写消息位置信息是单线程还是多线程？ 1](#_Toc364609621)

[4 Consumer拉消息是单个方式还是批量方式拉？ 1](#_Toc364609622)

[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？ 2](#_Toc364609623)

[6 异步刷盘方式，是否会导致内存爆掉？ 2](#_Toc364609624)

[7 Consumer拉消息是否使用了sendfile？ 2](#_Toc364609625)

[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？ 3](#_Toc364609626)

[9 PAGECACHE使用过多，是否会影响swap？ 3](#_Toc364609627)

[10 JVM CRASH后，写入PAGECACHE的未刷盘数据是否会丢失？ 3](#_Toc364609628)

[11 PAGECACHE映射总大小是否有限制？ 3](#_Toc364609629)

[12 是否可以认为Linux64位可以无限制映射PAGECACHE？ 4](#_Toc364609630)

[13 消息如何在java堆，物理内存，虚拟内存，磁盘之间流动？ 5](#_Toc364609631)

[14 Commit Log与Consume Queue使用long类型标识offset，是否会溢出？ 6](#_Toc364609632)

[15 RocketMQ Server关闭时，数据安全性如何保证？ 7](#_Toc364609633)

[16 RocketMQ Server重启时，如何Load数据？ 7](#_Toc364609634)

[17 RocketMQ如何区别是正常退出还是异常退出？ 8](#_Toc364609635)

[18 RocketMQ有哪些自我保护措施？ 8](#_Toc364609636)

[19 RocketMQ是否需要流控？ 8](#_Toc364609637)

[20 RocketMQ为什么可以支持海量（1万以上）队列？ 9](#_Toc364609638)

[21 Java中使用MapedByteBuffer可能会使JVM CRASH，RocketMQ如何避免？ 9](#_Toc364609639)

[22 订阅关系丢失，是否也会丢失消息？ 10](#_Toc364609640)

[23 存储层的刷盘策略是什么？ 10](#_Toc364609641)

[24 存储层视图？ 11](#_Toc364609642)

# 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。

# 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。

# 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。

# 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。

# 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

# 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

# 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）

# 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

# 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

# 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

# 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？

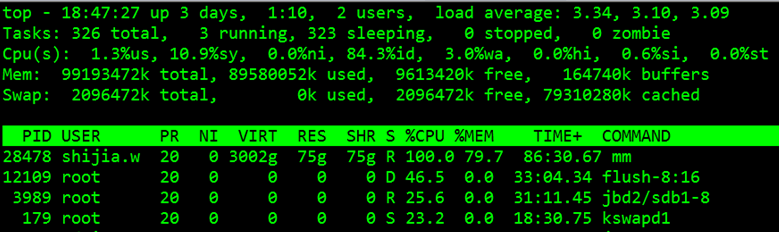
理论上，64位机器地址空间可认为无限大（但是实际由于cpu地址空间限制，可能会小于2的64次方）。

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>



# 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。

# 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。

* 1. Consumer拉消息（异常消费），消息直接从PAGECACHE（数据在虚拟内存）转入socket。
  2. 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。
  3. 同5一致。
  4. 同6一致。

# 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

# RocketMQ Server关闭时，数据安全性如何保证？

1. 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。
2. 异步刷盘情况下，异常关闭（重启时，必须要进行纠错）
   1. 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）

* 1. 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。

1. 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。

# RocketMQ Server重启时，如何Load数据？

1. 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。

1. 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。

# 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。

# 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。

# 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。

# 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，避免了磁盘竟争。

# 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。

# 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。

# 存储层的刷盘策略是什么？



图表 2刷盘策略

# 存储层视图？



图表 4存储层视图