


a417930422的专栏

目录视图

摘要视图

RSS 订阅


个人资料



nice2mitu

访问: 15001次

积分: 553

等级: 

排名: 千里之外

原创: 30篇

转载: 1篇

译文: 2篇

评论: 4条

文章搜索

文章分类

rocketmq设计与实现 (10)

rocketmq源码分析及注意事项 (12)

gc (1)

调优 (1)

java (6)

linux (5)

daemontools (1)

zookeeper (2)

流程 (1)

缓存 (1)

memcached (1)

文章存档

2016年09月 (10)

2016年05月 (1)

2016年04月 (1)

2016年03月 (1)

2016年02月 (12)

展开

阅读排行

rocketmq问题汇总-一个c (2281)

CMS GC日志详细分析 (1764)

【1024程序员节】获奖结果公布

【观点】有了深度学习，你还学传统机器学习算法么？

【资源库】火爆了的React Native都在研究什么

rocketmq3.26研究之一存储层

标签: [rocketmq](#)

2016-02-15 11:26 684人阅读 评论(0) 收藏 举报

分类:

java (5)

rocketmq源码分析及注意事项 (11)

版权声明：本文为博主原创文章，未经博主允许不得转载。

1. MappedFile

对MappedByteBuffer的封装，具有创建文件（使用非堆区内存），写入，提交，读取，释放，关闭等功能

关键字段解释：

1 fileFromOffset

单看这个字段很难明白什么意思，如果联系上MappedFileQueue来看的话，就能明白了，该字段代表了这个文件在queue中的偏移量，比如0，表示queue中的第一个文件，1024表示queue中的第二个文件（假设mappedFileSize为1024）。

重要的方法：

1 appendMessage(Object msg,AppendMessageCallback cb)

该方法通过回调AppendMessageCallback实现数据存储。

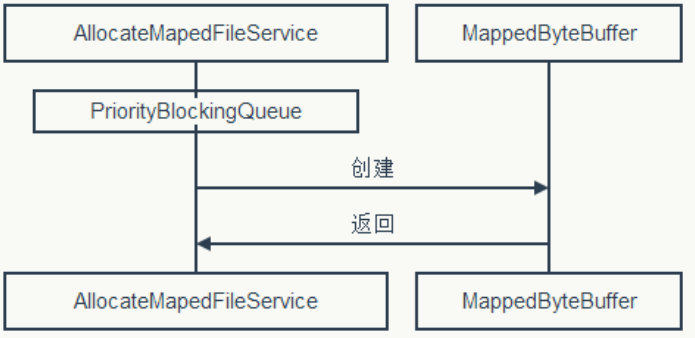
2 selectMappedBuffer(int pos, int size)

返回从pos到size的内存映射，用于读取数据。

2. AllocateMappedFileService

对并发创建MappedFile的请求，转化为串行化操作，并通过ConcurrentHashMap保证并发的对同一路径的创建请求只执行一次。

图示如下：



3. MappedFileQueue

顾名思义，该类代表了MappedFile组成的队列（由大小相同的多个文件构成）。

重要方法：

http://blog.csdn.net/a417930422/article/details/50606732

1/7

- json性能对比 fastjson jac (924)
- 低停顿互联网应用程序一 (714)
- rocketmq3.26研究之四Di (708)
- rocketmq3.26研究之一存 (684)
- rocketmq问题汇总-instar (677)
- rocketmq3.26研究之六Di (665)
- rocketmq3.26研究之五Di (493)
- 缓存失效时防止穿透DB的 (485)

评论排行

- rocketmq问题汇总-一个c (3)
- 低停顿互联网应用程序一 (1)
- memcache一键安装脚2 (0)
- json性能对比 fastjson jac (0)
- So you want to be a zool (0)
- 缓存失效时防止穿透DB的 (0)
- 线程安全的DateFormatU (0)
- daemontools监控zooke (0)
- 10.零拷贝原理 (0)
- 无锁编程初探 - 结果很 (0)

推荐文章

- * 2016 年最受欢迎的编程语言是什么？
- * Chromium扩展（Extension）的页面（Page）加载过程分析
- * Android Studio 2.2 来啦
- * 手把手教你做音乐播放器（二）技术原理与框架设计
- * JVM 性能调优实战之：使用阿里开源工具 TProfiler 在海量业务代码中精确定位性能代码

最新评论

- rocketmq问题汇总-一个consume nice2mitu: @fei33423:对，其实是rocketmq作者的开
- rocketmq问题汇总-一个consume 个人渣记录仅为自己搜索用: 明白了,你是在说 group配置一样,但是topic 配置不一样的两个consumer集群.
- rocketmq问题汇总-一个consume 个人渣记录仅为自己搜索用: 第一张图画错了. consumer1,consumer2都会订阅topic1,topic2. 导致...
- 低停顿互联网应用程序一步一步ifcs_our2009: 写得很详细，简单粗暴，赞一个，看了好多文章，都抓不到重点，泛泛而论。



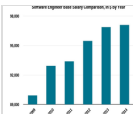
近视手术的危害



近视眼



前端工程师待遇



软件工程

3.1 getLastMappedFile

获取队列最后一个MappedFile对象，以下两种情况会创建新的MappedFile对象：

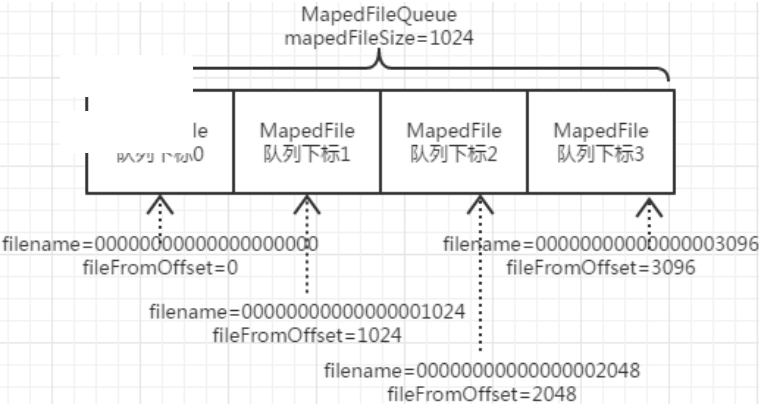
- 1 最后一个不存在
- 2 文件写满了

感觉并发调用该方法会导致创建多个MappedFile

3.2 findMappedFileByOffset(final long offset)

根据offset/filesize计算该offset所在那个文件中。

为了理解findMappedFileByOffset，我们假设每个文件的大小是1024K，参考以下图示：



如果现在想查找3021在那个文件中，可以按如下计算：

(3021-0)/1024=2 即可知其所在队列下标为2的MappedFile中

释义如下：(offset-第一个文件的fileFromOffset)/mappedFileSize

4. CommitLog.GroupCommitService

同步刷盘服务。每提交一条消息，就触发该服务，消息写入后立即刷盘，会导致写入线程阻塞。其持有一个GroupCommitRequest的队列。GroupCommitRequest.nextOffset指明这个消息写入到了那个位置。GroupCommitService根据提交的位置是否落后于这个消息的位置进行刷盘。

5. CommitLog.FlushRealTimeService

实时刷盘服务（异步刷盘）。默认1秒刷一次或来消息刷一次。一次至少刷4个page.一个page为4k.

6. CommitLog

用于存储消息的抽象封装，内部采用MappedFileQueue实现了消息文件队列功能。

重要字段

HashMap topicQueueTable

用于记录某个topic在某个queueId共写入了多少个消息，put一个消息加1.

重要方法：

1 putMessage(final MessageExtBrokerInner msg)

存储消息。

主要分3步：**查找文件(getLastMappedFile)**，**写入数据(DefaultAppendMessageCallback)**，**刷盘(FlushRealTimeService)**

最终产生实际存储消息的队列文件如下：

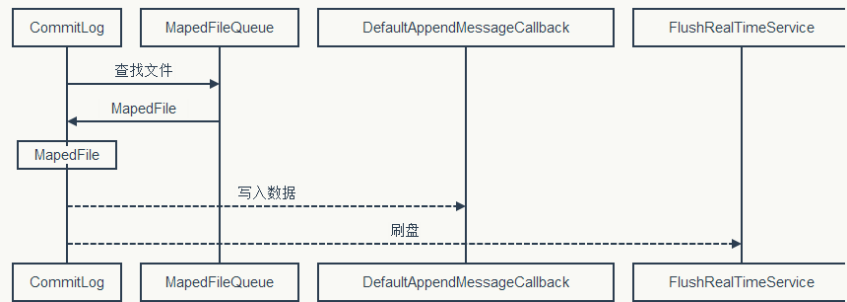
\${storePathRootDir}/commitlog/消息队列文件

消息队列文件名规则：

文件默认大小(1024*1024*1024)依次递增，类似如下：

00000000000000000000
00000000001073741824
00000000002147483648
...

整个流程如下图所示：

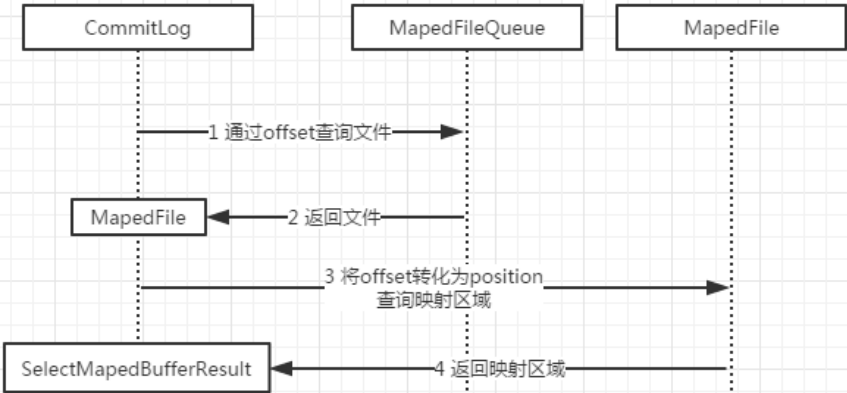


2 getMessage(final long offset, final int size)

offset:绝对偏移量，可以用其调用findMappedFileByOffset查询MappedFile

size:欲查询的数据大小

其调用过程如下图：



7. CommitLog.DefaultAppendMessageCallback

该类主要用于完成消息的实际存储。

- 重要字段：
 - 1 maxMessageSize：消息的最大大小，默认512k
 - 2 msgIdMemory: 用于存储消息id=ip(4byte)+port(int)+queue中的偏移量(long)=16byte
 - 3msgStoreItemMemory:用于存储消息，大小为maxMessageSize+8
 - 重要方法：
 - doAppend(long fileFromOffset, ByteBuffer byteBuffer, int maxBlank, Object msg)
 - fileFromOffset：queue中的偏移量
 - byteBuffer：虚拟内存映射空间
 - maxBlank：该文件剩余的空间
 - msg：消息
- 该方法会实际的写入数据，并生成消息id。

8. DefaultMessageStore.DispatchMessageService

CommitLog.putMessage时，会分发消息的位置等信息，主要用于生成消息索引和消息的消费队列(会调用DefaultMessageStore.putMessagePostionInfo进行消费队列消息位置写入)。

9. DispatchRequest

消息位置分发的对象封装。

- 重要字段
 - 1 String topic: 主题
 - 2 int queueId: 逻辑队列id
 - 3 long commitLogOffset: commit数据文件中该消息写入位置的起始偏移量
 - 4 msgSize: 消息长度
 - 5 consumeQueueOffset: 消费队列的大小

10. IndexService

消息索引服务，主要根据topic和keys(producer设置)建立索引，并提供查询功能。

11. ConsumeQueue

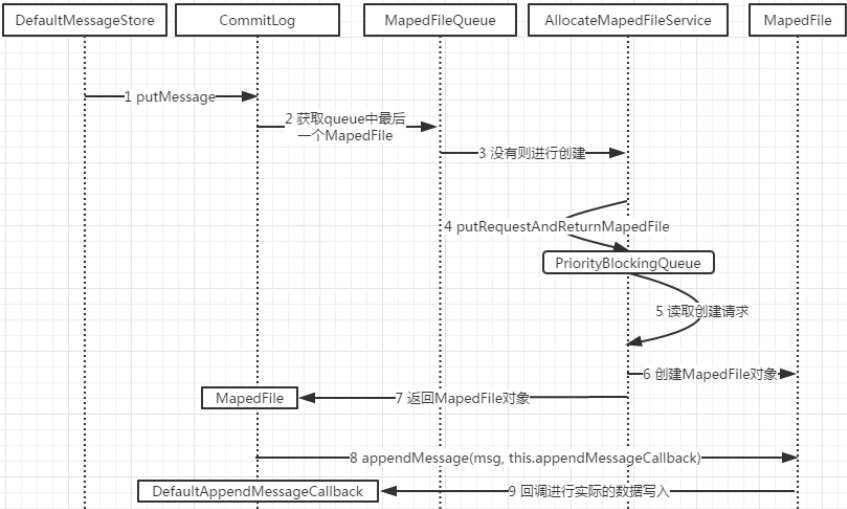
消费队列的实现，该消费队列主要存储了消息在CommitLog的位置，与CommitLog类似，内部采用MappedFileQueue实现了消息位置文件队列功能
一个topic和一个queueId对应一个ConsumeQueue。
默认queue存储30W条消息。
每个消息大小为20个字节,详细如下:
offset(long 8字节)+size(int 4字节)+tagsCode(long 8字节)

- 重要方法：
1 putMessagePostionInfo(long offset, //消息在commitLog中的起始位置
int size, //消息长度
long tagsCode,
long cqOffset//该消息在topic对应的queue中的下标,参照Commitlog.topicQueueTable
)
该方法主要实现了消息位置的存储，并产生消息文件：
storePathRootDir/consumequeue/{topic}/{queueId}/消息位置队列文件
消费队列文件名规则：
消息数(30W)*消息位置固定大小(20字节)=6000000字节
故每6000000字节一个文件，文件名依次递增，前缀不够20位补0，类似如下：
00000000000000000000
000000000000006000000
00000000000012000000
...
2getIndexBuffer(long startIndex)
startIndex代表了起始偏移量索引。
该方法会根据startIndex查找到相应的索引文件，并返回该文件当前写到的位置到startIndex的消息分区。
startIndex为客户端目前消费的进度，实际为消息位置队列文件的消费偏移量索引。

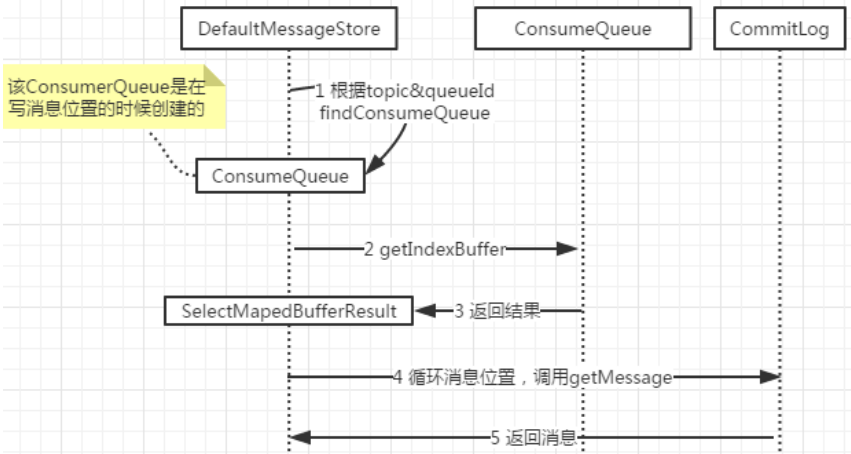
12. DefaultMessageStore

该类实现了消息存储，消息位置存储，读取，索引，HA等功能

- 消息存储功能
putMessage(MessageExtBrokerInner msg)
该方法会进行一些列校验，是否为slave，可写，topic长度等，之后调用CommitLog进行消息存储，并调用StoreStatsService进行调用统计。
好了，到这里，我们需要一张综合的图来展示调用的过程：



- 消息位置存储
putMessagePostionInfo(String topic, //主题, 后续章节解释
int queueId, //队列id, 后续章节解释
long offset, //commit log中消息存储的起始偏移量
int size, //消息的大小
long tagsCode,
long storeTimestamp,
long logicOffset)//topic+queueId消费队列中有都少条消息
该方法会调用ConsumeQueue.putMessagePostionInfo把消息位置(在CommitLog中写入的消息)写入到消费队列。
- 消息读取功能
getMessage(String group, //即ConsumerGroup在后续章节中解释
String topic, //主题, 后续章节解释
int queueId, //队列id, 后续章节解释
long offset, //偏移量
int maxMsgNums, //最大拉取的消息个数
SubscriptionData subscriptionData) //消息过滤实现
该方法用于消息读取, 此处涉及到了两个队列, 即消息队列CommitLog和消息位置队列ConsumeQueue, 为了阐明该方法的逻辑, 参考如下图:

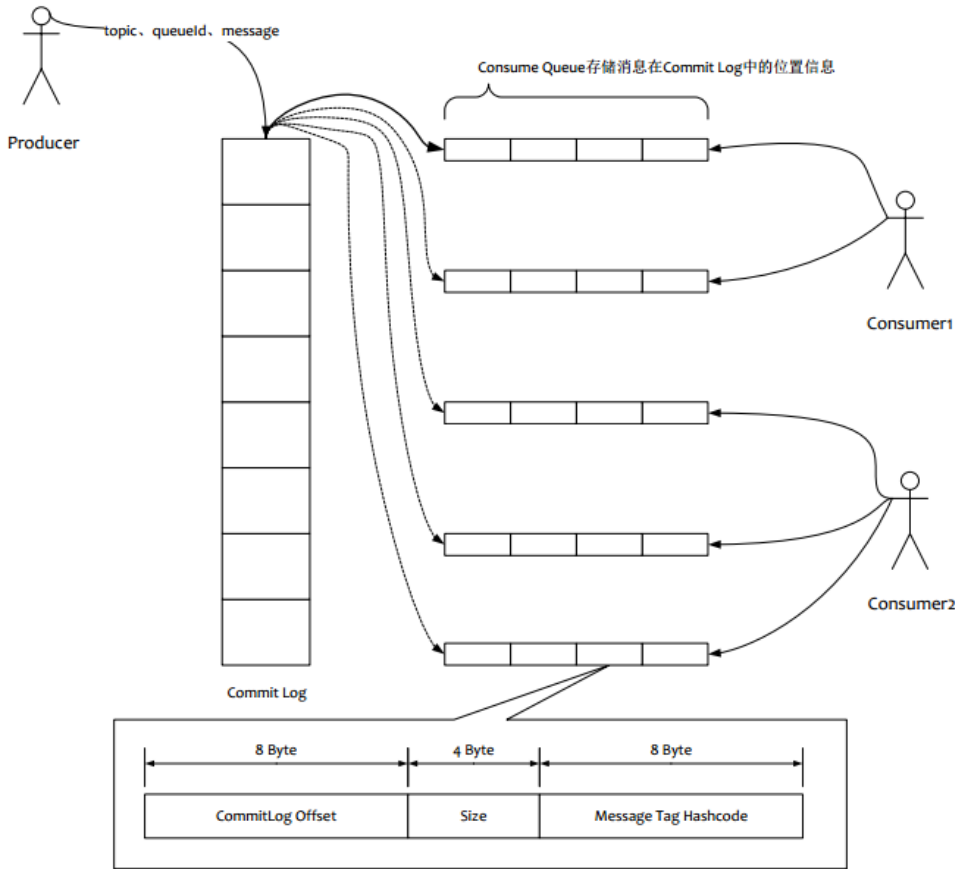


13. HAService

负责主从之间的同步双写, 异步复制

- 重要字段:
 - 1 HAPort: 默认采用broker启动时监听的端口+1来作为HA端口
 - 2 haMasterAddress: 默认如果不设置的话从NameServer来获取, 而NameServer上的是broker启动的时候注册上去的, 默认为brokerIP2+HAPort, brokerIP2默认值为该broker启动地址。
故对于一个slave的broker来说, 其向NameServer注册时会根据BrokerName拉取master地址和haMasterAddress地址。
- 重要方法:
 - 1 主从同步双写如何实现?
CommitLog.putMessage存储消息时, 如果发现配置为SYNC_MASTER, 则会调用HAService.GroupTransferService.putRequest(GroupCommitRequest request)来进行消息的同步双写, 但是其实此方法只是检测同步结果, 看看是否消息已经写到的slave, 真正实现通信的是HACConnection。
对于broker来说, 不管是master还是slave都会启动HAService.AcceptSocketService(其内部对于到来的链接使用HACConnection来处理), 该服务的主要作用就是在HAPort进行监听, 响应slave的应答, 并向slave push数据。
 - 2 异步复制如何实现?
HAService会启动HAService.HAClient用于slave从master同步数据。

14. 最后用rocketmq作者的一张图总结一下：



从上图可知：
ComimitLog是存储消息的文件，多个文件构成一个队列。
而ConsumerQueue是消息的位置文件，多个文件同样构成一个队列。
这样，所有的生产者产生的消息都会顺序写入到ComimitLog文件中。
而消费者只消费自己对应的topic+queue的文件，这样不会因为消费队列的增加导致磁盘io问题。

顶 踩
0 0

上一篇 [rocketmq3.26研究之六DefaultMQPushConsumer消费流程](#)
下一篇 [rocketmq3.26研究之三NameServer](#)

我的同类文章

java（5）		rocketmq源码分析及注意事项（11）	
• ip-int-byte[]互转-使用java原...	2016-05-11 阅读 110	• json性能对比 fastjson json	2014-02-26 阅读 926
• 缓存失效时防止穿透DB的策略	2014-01-13 阅读 485	• 线程安全的DateFormatUtil	2014-01-09 阅读 411
• 低停顿互联网应用程序一步...	2013-11-25 阅读 714		

猜你在找

- 360度解析亚马逊AWS数据存储服务
- Android之数据存储
- iOS开发高级专题—数据存储
- RocketMQ原理解析-broker 2消息存储
- rocketmq的存储数据结构
- RocketMQ原理解析-Remoting3通信层整体交互图

顾荣: 开源大数据存储系统Alluxio (原Tachyon) 的原 解决了Cocoapods Undefined symbols for
1. 16. ARM裸机第十六部分-shell原理和问答机制引入 RocketMQ原理解析-Remoting2 通信层底层传输协议




查看评论

暂无评论

发表评论

用户名: chenyongsuda

评论内容:



提交

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

- 全部主题
- Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
- VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
- BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
- Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC
- coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
- Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
- Angular Cloud Foundry Redis Scala Django Bootstrap