**BlotMQ-Store技术文档说明**

针对版本V1.0.0

**©成都基础平台架构**

**2017/11/21**

BoltMQ-Store修订记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 修订内容 | 作者 | 审核 | 修订日期 |
| V1.0.0 | 初始版本 | 周非 | 基础平台架构组 | 2017/11/21 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目 录

[1 存储 4](#_Toc499223774)

[1.1 概述 4](#_Toc499223775)

[1.2 零拷贝技术 4](#_Toc499223776)

[1.3 CommitLog 4](#_Toc499223777)

[1.4 ConsumeQueue 7](#_Toc499223778)

[1.5 索引 9](#_Toc499223779)

[1.6 主从同步 11](#_Toc499223780)

[1.7 刷盘 13](#_Toc499223781)

[1.8 文件清理 13](#_Toc499223782)

[附件一BoltMQ开发者联系方式 15](#_Toc499223783)

# 存储

## 概述

存储模块主要包含存储Producer生产的消息、ConsumeQueue、索引等数据以及主从同步、刷盘、清理服务等。

## 零拷贝技术

零拷贝是通过将文件映射到内存上，直接操作文件，相比于传统的io(首先要调用系统IO，然后要将数据从内核空间传输到用户空间)，避免了很多不必要的数据拷贝，提高存储性能。存储消息，使用了零拷贝，零拷贝包含以下两种方式：

|  |  |  |
| --- | --- | --- |
| 方式 | 优点 | 缺点 |
| mmap + write | 即使频繁调用，使用小块文件传输，效率也很高 | 不能很好的利用 DMA 方式，会比 sendfile 多消耗 CPU，内存安全性控制复杂 |
| sendfile | 可以利用 DMA 方式，消耗 CPU 较少，大块文件传输效率高，无内存安全新问题 | 小块文件效率低于 mmap 方式 |

BlotMQ采用mmap+write方式，因为有小块数据传输的需求，效果会比sendfile更好。

## CommitLog

CommitLog用于存储真实消息数据。CommitLog路径默认为用户工作目录/store/commitlog。

**CommitLog存储目录结构：**

commitlog

- 00000000000000000000

- 00000000001073741824

**commitlog文件名生成的规则：**

文件名的长度为20位，左边补零，剩余的为文件起始偏移量（第一个文件起始偏移量为0）；

文件名字根据指定commitlog文件大小（默认文件大小为1G，可以通过MessageStoreConfig的mapedFileSizieCommitLog进行配置）递增，文件大小单位为字节。

例如：

默认commitlog文件大小为1G=1073741824b

第一文件的起始偏移量为0，不足20位进行补零，故文件名00000000000000000000，当第 一文件写满，第二文件的起始偏移量为1073741824，不足20位进行补零，故文件名为00000000001073741824，后面的文件名以此类推。

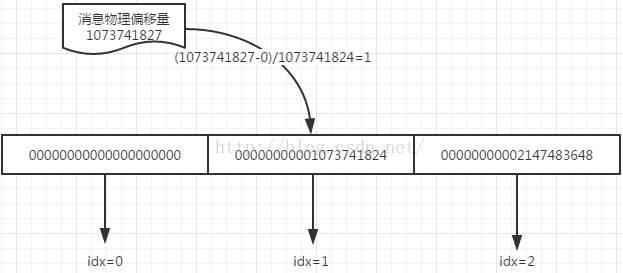
文件n起始偏移量 = size \* (n- 1)

文件1起始偏移量 = 1073741824 \* (1 - 1) = 0

文件2起始偏移量 = 1073741824 \* (2 - 1) = 1073741824

通过commitlog文件名能够方便快速定位信息所在的文件。

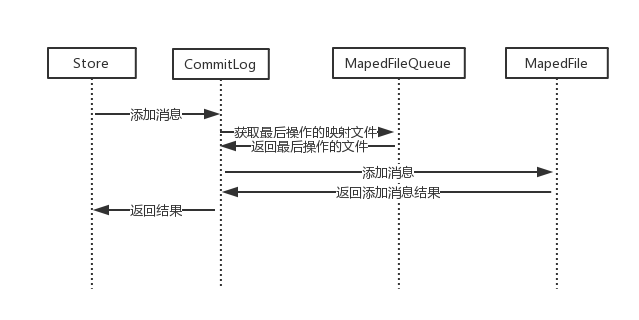
文件Index = (消息的起始物理偏移量-最早的文件的起始偏移量)/文件大小，即 (1073741827-0)/1073741824=1，可得知该消息在队列中的第二个文件中：



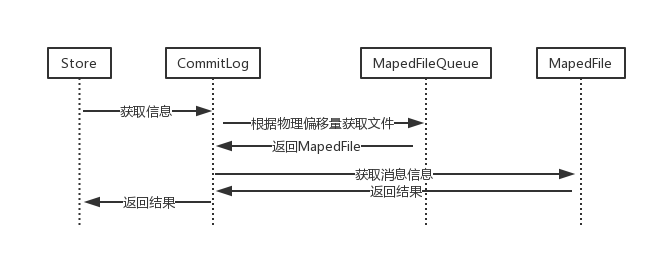
**commitlog文件的消息结构：**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **字段** | **说明** | **字节数** | **备注** |
| 1 | TotalSize | 消息总长度 | 4 |  |
| 2 | MagicCode | MagicCode | 4 | MagicCode分为：MessageMagicCode、  BlankMagicCode。MessageMagicCode表示正确的消息内容；BlankMagicCode表示CommitLog文件空间不足，采用空字节占位写满文件。 |
| 3 | BodyCRC | 消息内容CRC | 4 | BodyCRC的值是对消息内容（body）进行CRC32生成的32bit冗余校验码，用于确保消息的正确性。 |
| 4 | QueueId | 消息队列编号 | 4 |  |
| 5 | Flag | 消息标志 | 4 |  |
| 6 | QueueOffset | 消息队列位置 | 8 | 自增值，消息队列逻辑位置，通过该值才能查找到consume queue中的数据；QueueOffset \* 20才是消息队列的物理偏移量。 |
| 7 | PhysicalOffset | 物理位置 | 8 |  |
| 8 | SysFlag | MessageSysFlag | 4 |  |
| 9 | BornTimestamp | 生产消息时间戳 | 8 |  |
| 10 | BornHost | 生产消息的地址+端口 | 8 |  |
| 11 | StoreTimestamp | 存储消息时间戳 | 8 |  |
| 12 | StoreHost | 存储消息的地址+端口 | 8 |  |
| 13 | ReconsumeTimes | 重新消费消息次数 | 4 |  |
| 14 | PreparedTransationOffset |  | 8 |  |
| 15 | BodyLength | 消息内容长度 | 4 |  |
| 16 | Body | 消息内容 | bodyLength |  |
| 17 | TopicLength | Topic长度 | 1 |  |
| 18 | Topic | topic | topicLength |  |
| 19 | PropertiesLength | 附加属性长度 | 2 |  |
| 20 | Properties | 附加属性 | propertiesLength |  |

添加CommitLog数据，将数据写入到MapedFile，每个MapedFile对应着一个储存消息的二进制文件，MapedFile在创建时会映射到内存上，添加消息时将需要保存的数据写入内存，后续有刷盘服务会将内存中数据持久化到二进制物理文件中，下图是添加CommitLog数据的主要业务流程：



查询CommitLog数据，直接从映射的内存中根据物理偏移量以及数据大小，获取数据，下图是查询CommitLog数据的主要业务流程：



## ConsumeQueue

消费者逻辑队列，对应/store/consumequeue文件夹，每个消费队列文件目录机构如下：

consumequeue

-- topic

-- queue id

-- 00000000000000000000

-- 00000000000000001040

-- 00000000000000002080

**consumequeue文件名生成规则：**

commitlog文件名生成规则一致，需要注意的是：maped文件大小为=向上取整（指定size/消息位置信息size）\* 消息位置信息size

例如：

指定消费队列文件大小=1024

消息位置信息size = 20

mapedFileSize = 向上取整（1024 / 20) \* 20

mapedFileSize = 1040

**consumequeue文件结构：**

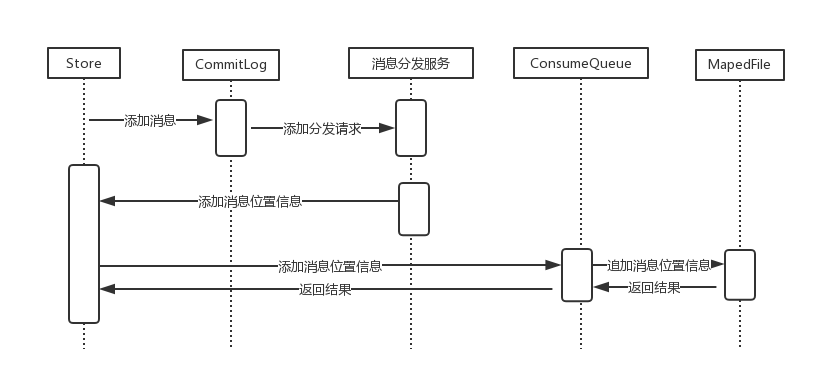
ConsumeQueue中并不需要存储消息的内容，而存储的是消息在CommitLog中的offset。也就是说，ConsumeQuue其实是CommitLog的一个索引文件。



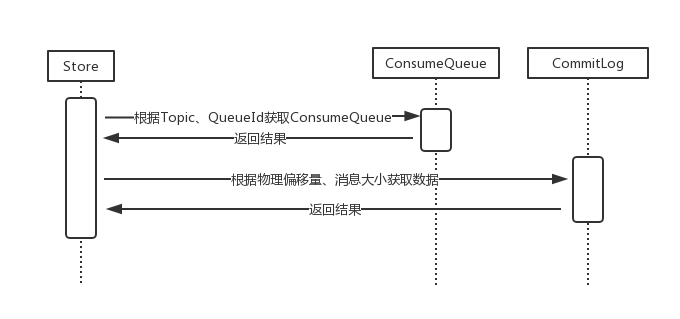
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **字段** | **说明** | **字节数** | **备注** |
| 1 | CommitLog Offset | CommitLog的起始物理偏移量physical offset | 8 |  |
| 2 | Size | 消息的大小 | 4 |  |
| 3 | Message Tag Hashcode | 消息Tag的哈希值 | 8 | 用于订阅时消息过滤（订阅时如果指定  了Tag，会根据HashCode来快速查找到  订阅的消息） |

ConsumeQueue是定长的结构，每条数据大小为20个字节，每个文件默认大小为600万个字节。Consumer消费消息的时候，需要2个步骤：首先读取ConsumeQueue得到offset，然后读取CommitLog得到消息内容。

添加消息时，添加消息到commitLog后会向分发服务添加一个分发请求，分发服务调用MessageStore添加消息位置信息，根据消息的Topic、QueueId获取ConsumeQueue，消息的位置信息追加到对应的消费队列中，最终保存的二进制文件中，主要流程如下图：



获取消息时， 首先根据Topic、QueueId获取ConsumeQueue，然后根据消息逻辑offset，获取消息的物理偏移量、消息的Size，最后根据消息的物理偏移量、消息的Size获取CommitLog数据，主要流程如下图：



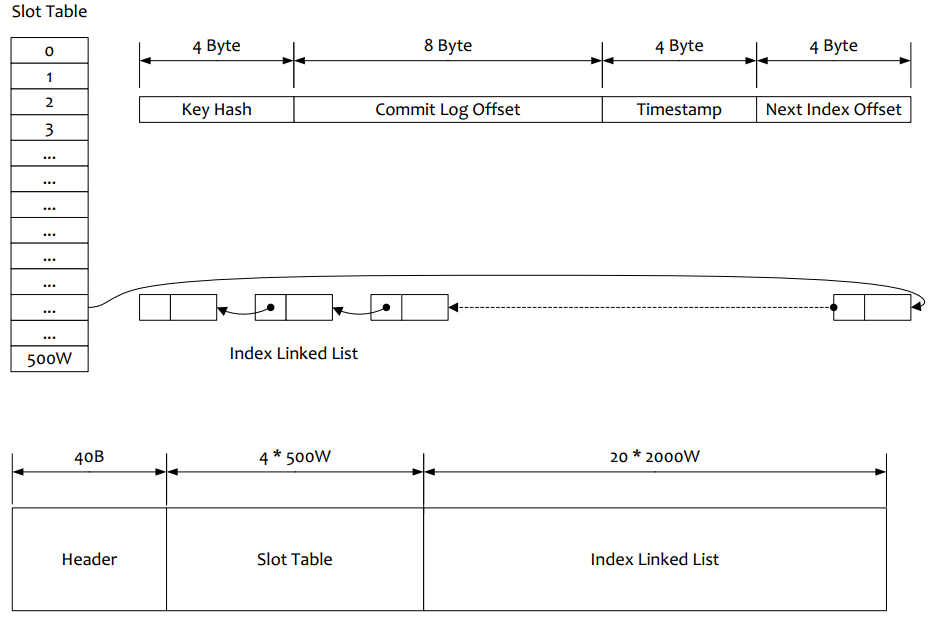
## 索引

**IndexService（索引服务）**

IndexService用于创建索引文件集合，当用户想要查询某个topic下某个key的消息时，能够快速响应。

**Index File（索引文件）**

IndexFile存储消息索引的文件，文件结构如下：



索引文件由三个部分组成：Header（索引文件头信息）、Slot Table（槽位信息）、Index Linked List（消息的索引内容）

Index Header：索引文件头信息由40个字节的数据组成。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **字段** | **说明** | **字节数** | **备注** |
| 1 | BeginTimestamp | 索引文件开始时间 | 8 | 第一个索引创建的时间 |
| 2 | EndTimestamp | 索引文件结束时间 | 8 | 最后一个索引创建的时间 |
| 3 | BeginPhyOffset | 索引文件开始的物理偏  移量 | 8 | 第一个索引对应的CommitLog物理偏移量 |
| 4 | EndPhyOffset | 索引文件结束的物理偏  移量 | 8 | 最后一个索引对应的CommitLog物理偏移  量 |
| 5 | HashSlotCount | 索引文件占用的槽位数 | 4 |  |
| 6 | IndexCount | 索引的个数 | 4 |  |

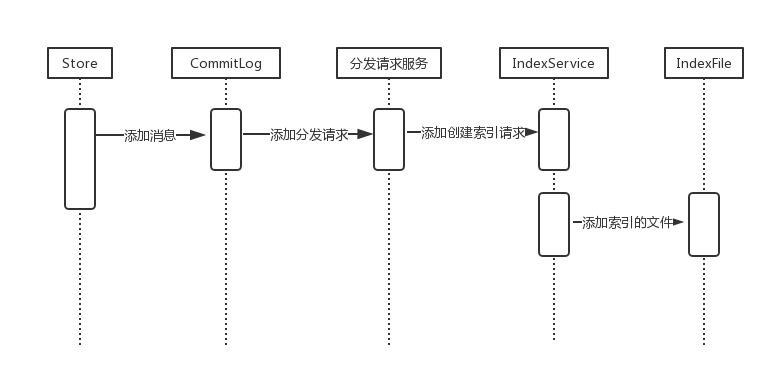
Slot Table

Index Linked List：消息的索引内容链表，默认每个文件有2000W消息索引内容组成，每个消息索引内容为20个字节的数据。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **字段** | **说明** | **字节数** | **备注** |
| 1 | KeyHash | key的哈希值 | 4 | topic-key(key是消息的key)的hashCode组成 |
| 2 | PhyOffset | commitLog的物理偏移量 | 8 |  |
| 3 | Timestamp | 索引创建的时间 | 4 |  |
| 4 | NextIndexOffset | 下一个索引的索引地址 | 4 |  |

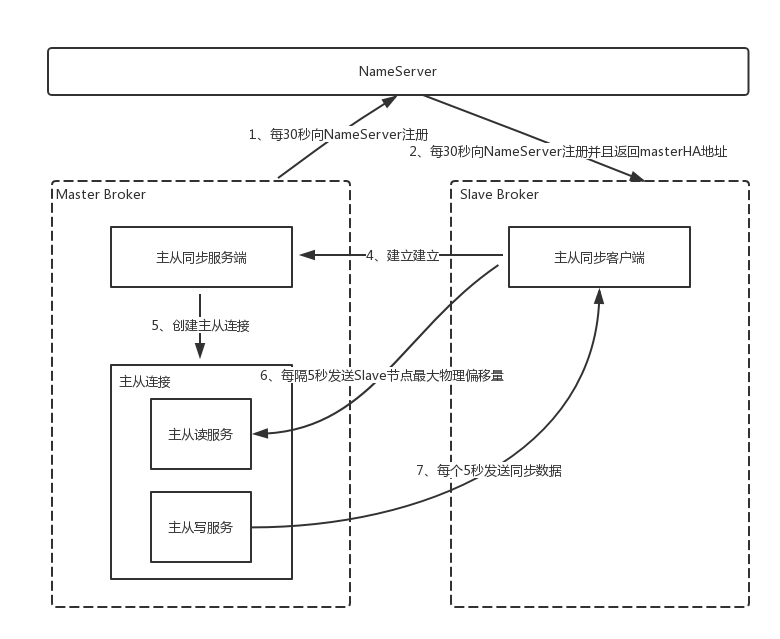
IndexFile的创建过程：

首先在DispatchMessageService写入ConsumeQueue后，会再调用indexService.putRequest，添加索引请求；IndexService定时获取创建索引请求，调用IndexService的buildIndex进行创建索引。



## 主从同步

在集群模式的部署方式中，Master与Slave配对是通过指定相同的brokerName参数来配对，Master的BrokerId必须是0，Slave的BrokerId必须是大于0的数。一个Master下面可以挂载多个Slave，同一个Master下的多个Slave通过指定不同的BrokerId来区分。



**主从同步服务**

存储模块启动时，会启动主从同步服务，主从同步服务主要的组成部分是：主从同步服务端、主从同步客户端

**主从同步服务端**

接收slave节点的连接请求，接收到请求后会建立主从连接，接受和传递主从之间数据。

**主从连接**

主从连接主要由主从写服务、主从读服务组成，主从写服务主要用于master传输同步数据，主从读服务主要用于接受slave节点发送的offset信息。

主从写服务传输的数据结构：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **字段** | **说明** | **字节数** | **备注** |
| 1 | Offset | commitLog物理偏移量 | 8 | 同步commitLog物理偏移量 |
| 2 | BodySize | 传输数据的大小 | 4 |  |
| 3 | BodyData | 传输数据的内容 | BodyLength |  |

**主从同步客户端**

连接master节点，定时上报offset以及接收master节点传输的同步数据。

主从同步客户端上报的数据结构：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **字段** | **说明** | **字节数** | **备注** |
| 1 | Offset | commitLog物理偏移量 | 8 | slave节点的最大commitLog物理偏移量 |

主从同步客户端上报offset时，会获取当前最大CommitLog文件物理偏移量。如果HAClient是首次上报offset，并且上报的offset为0，master节点会获取最后一个CommitLog文件进行传输，其余的CommitLog文件不会进行同步。上报的offset不为0，master节点会从上报的offset进行同步。

## 刷盘

RocketMQ刷盘有两种方式，分为：同步刷盘、异步刷盘。

同步刷盘：在消息到达MQ后，RocketMQ需要将数据持久化，同步刷盘是指数据到达内存之后，必须刷到commitlog日志之后才算成功，然后返回producer数据已经发送成功。

异步刷盘：数据到达内存之后,返回producer说数据已经发送成功，然后再写入commitlog日志。

RocketMQ默认是使用异步刷盘。

逻辑队列刷盘服务(FlushConsumeQueueService)：用于将ConsumeQueue的File文件写入入里磁盘，

首先判断是否到达了刷盘时间，如果到达了，那么全盘通刷；否则，遍历所有的ConsumeQueue，调用cq.commit(flushConsumeQueueLeastPages)进行刷盘，flushConsumeQueueLeastPages是目前文件的未刷盘大小达到flushConsumeQueueLeastPages\*OS\_PAGE\_SIZE(1024\*4)个，才进行刷盘。

逻辑队列刷盘服务：定时将ConsumeQueue的数据从内存写入到文件。

## 文件清理

存储服务启动时，会启动定时清理文件服务，定时清除服务会每分钟定时清理CommitLog、ConsumeQueue文件。

**清理CommitLog文件服务**

清理CommitLog文件，需要满足以下任意一条件：

1、消息文件过期（默认72小时），且到达清理时点（默认是凌晨4点），删除过期文件。

2、消息文件过期（默认72小时），且磁盘空间达到了水位线（默认75%），删除过期文件。

3、磁盘已经达到必须释放的上限（85%水位线）的时候，则开始批量清理文件（无论是否过期），直到空间充足。

注：若磁盘空间达到危险水位线（默认90%），出于保护自身的目的，broker会拒绝写入服务。

**清理ConsumeQueue文件服务**

定时清理小于最小CommitLog物理偏移量的ConsumeQueue的文件。

# 附件一BoltMQ开发者联系方式

|  |  |  |
| --- | --- | --- |
| 姓名 | 联系方式 | 更新日期 |
| 郜焱磊 | **gyl\_adaihao@163.com** | 2017/11/21 |
| 田玉粮 | **idistyl@gmail.com** | 2017/11/21 |
| 尹同强 | [**tongqiangyin@gmail.com**](mailto:tongqiangyin@gmail.com) | 2017/11/21 |
| 罗继 | [**gunsluo@gmail.com**](mailto:gunsluo@gmail.com) | 2017/11/21 |
| 周飞 | **he236555699@163.com** | 2017/11/21 |
| 戎志宏 | **hzrong007@163.com** | 2017/11/21 |