RocketMQ-MQBroker

yongping.ren@ygomi.com

MQBroker

- Broker工作概述
- AllocateRequestService工作原理
- 消息处理机制

Broker工作概述：主要包含broker的启动过程，初始化工作，以及broker启动需要做的一些前期准备工作。

AllocateRequestService工作原理：主要涉及broker对消息存储的一些前期准备工作，以及broker如何实现对消息的高效读写的。

消息处理机制：指broker收到消息后是如何对消息进行处理的。

涉及知识点

- Netty
- NIO
- mmap
- 多线程，异步编程，锁

Netty: 基于事件的异步网络编程框架

NIO: 异步IO，相对于之前的BIO

mmap: 内存映射

多线程，异步编程，锁：自定义线程池，回调编程，RetreenLock，SpinLock，FileLock…

Netty

- 由JBOSS提供的一个Java开源框架，提供异步的、事件驱动的网络应用程序框架和工具，用以快速开发高性能、高可靠性的网络服务器和客户端程序。
- 高吞吐量，低延迟
- 低资源损耗
- 最小化不必要的内存拷贝

NIO

- 同步非阻塞IO
- Channel
- Buffer
- 多路复用器（selector，poll，epoll）

BIO：同步阻塞IO

NIO优点：

通过Channel注册到多路复用器上的状态来实现一种客户端与服务端的通信。

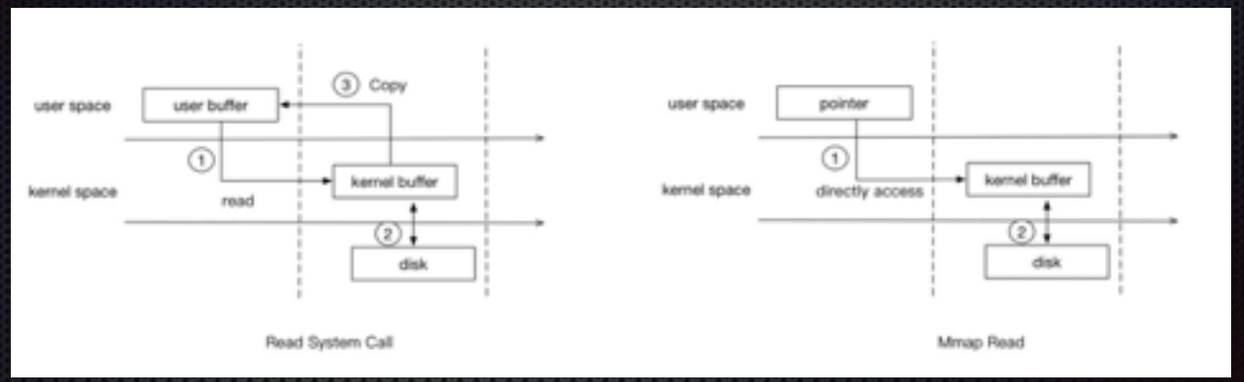
Channel中数据的读取是通过Buffer , 一种非阻塞的读取方式。

Selector 多路复用器 单线程模型， 线程的资源开销相对比较小

mmap

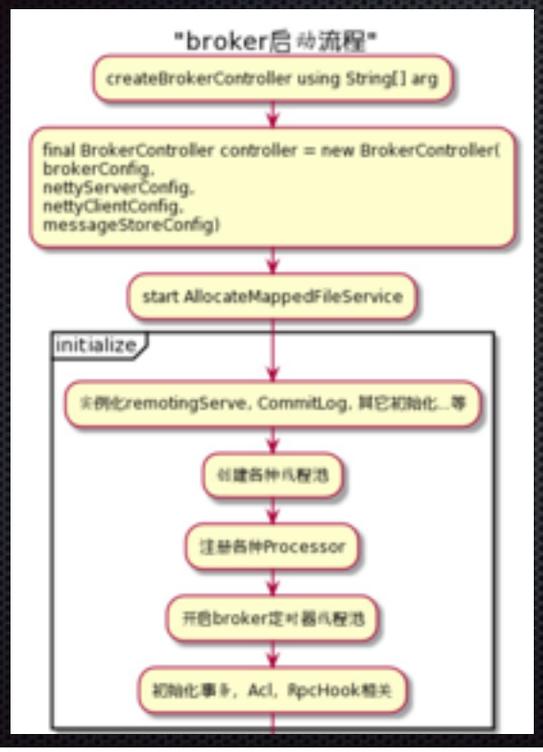
- 内存映射，减少传统IO将磁盘文件数据在操作系统内核地址空间的缓冲区和用户应用程序地址空间的缓冲区之间来回进行拷贝的性能开销

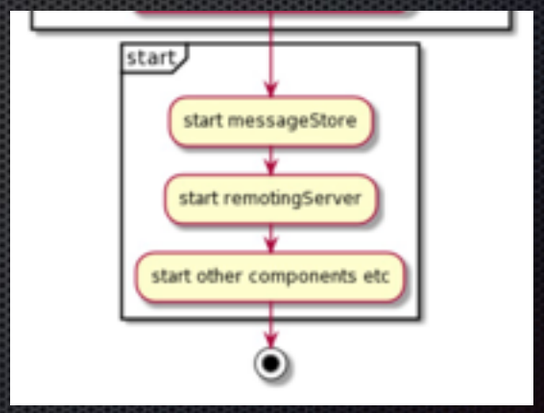
传统read与mmap比较



Broker工作概述

Broker启动流程


"broker启动流程"
createBrokerController using String[] arg
final BrokerController controller = new BrokerController(
brokerConfig,
nettyServerConfig,
nettyClientConfig,
messageStoreConfig)
start AllocateMappedFileService
initialize
实例化remotingServe, CommitLog, 其它初始化...等
创建各种线程池
注册各种Processor
开启broker定时器线程池
初始化事务，Acl，RpcHook相关


start
start messageStore
start remotingServer
start other components etc

- 分析输入参数，生成BrokerController实例
- 启动AllocateMappedFileService
- 实例化通信模块以及存储模块
- 创建各种线程池，如：heartbeatExecutor, consumerManageExecutor…等等
- 注册各种Processor，如：SendMessageProcessor
- 开启定时器任务，如：持久化consumerOffset，持久化topic filter信息等。
- 启动各种服务线程，包括：messageStore，remotingServer等。

NettyRemotingServer

- `private final ServerBootstrap serverBootstrap;`
- `protected final NettyEventExecutor nettyEventExecutor = new NettyEventExecutor();`
- `HashMap<Integer/* request code */, Pair<NettyRequestProcessor, ExecutorService>> processorTable`
- `private NettyServerHandler serverHandler;`

ServerBootstrap: Netty服务端入口，生成一个Netty服务器。

NettyEventExecutor：Netty event处理线程，包含一个event table，轮询处理收到的event事件，包括处理channel注册，下线等。

processorTable：保存各种processor实例，根据request code可以获取不同processor实例进行处理

NettyServerHandler：包含channelRead0回调函数，当channel收到消息后回调此函数

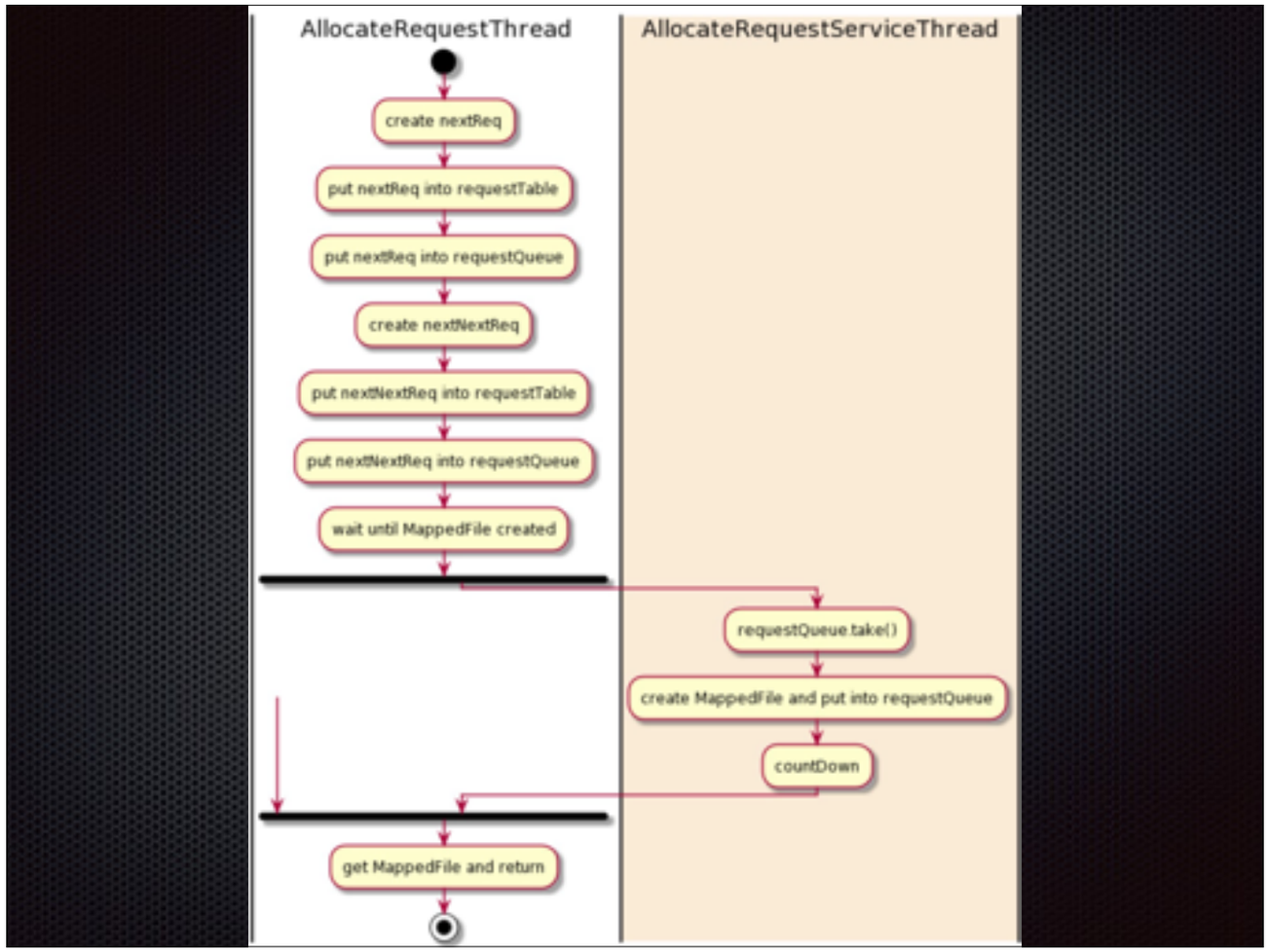
- 实例化各种NettyEventHandler，包括NettyEncoder，NettyConnectManageHandler，NettyServerHandler等。
- 实例化ServerBootstrap，并将各种handler注册进去
- 启动NettyServer
- 开启NettyEventExecutor线程，处理NettyEvent

AllocateRequestService

关键成员变量及方法

- ConcurrentMap<String/**file path**/, AllocateRequest> requestTable
- PriorityBlockingQueue<AllocateRequest> requestQueue
- public MappedFile putRequestAndReturnMappedFile(String nextFilePath, String nextNextFilePath, int fileSize)
- private boolean mmapOperation()

ConcurrentMap的value和PriorityBlockingQueue中的AllocateRequest保存的是同一份引用。



public MappedFile putRequestAndReturnMappedFile(String nextFilePath, String nextNextFilePath, int fileSize)
申请创建一个新的MappedFile，将生成两个request放入队列。
private boolean mmapOperation()
阻塞方法，一旦有request放入队列，将生成一个新的MappedFile，并更新队列中关于MappedFile的引用。

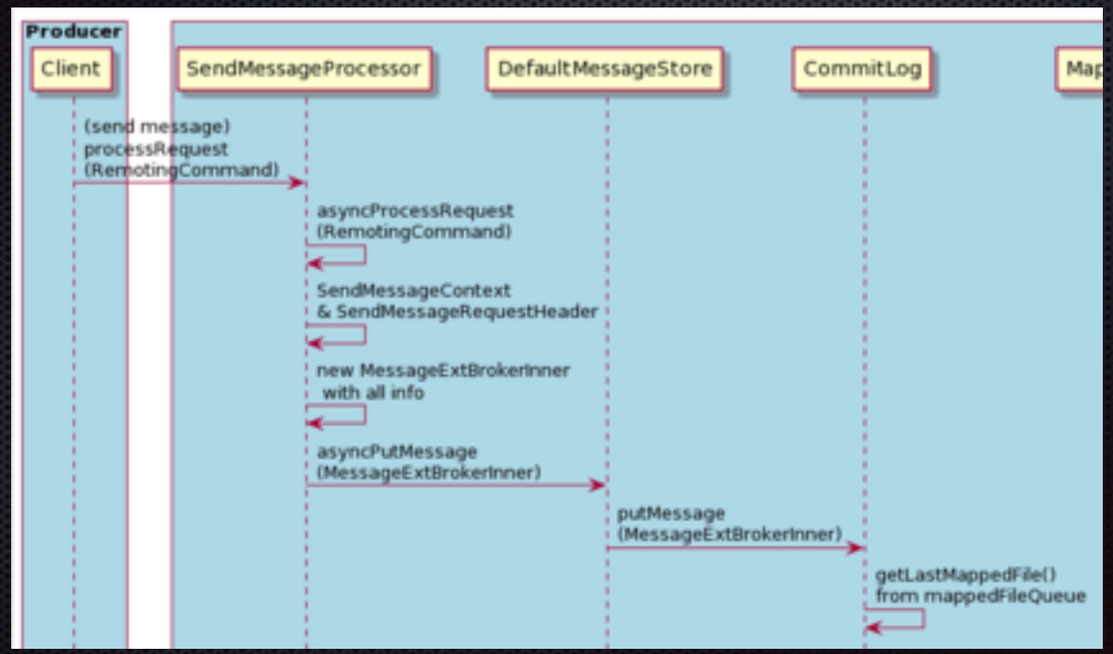
消息处理机制

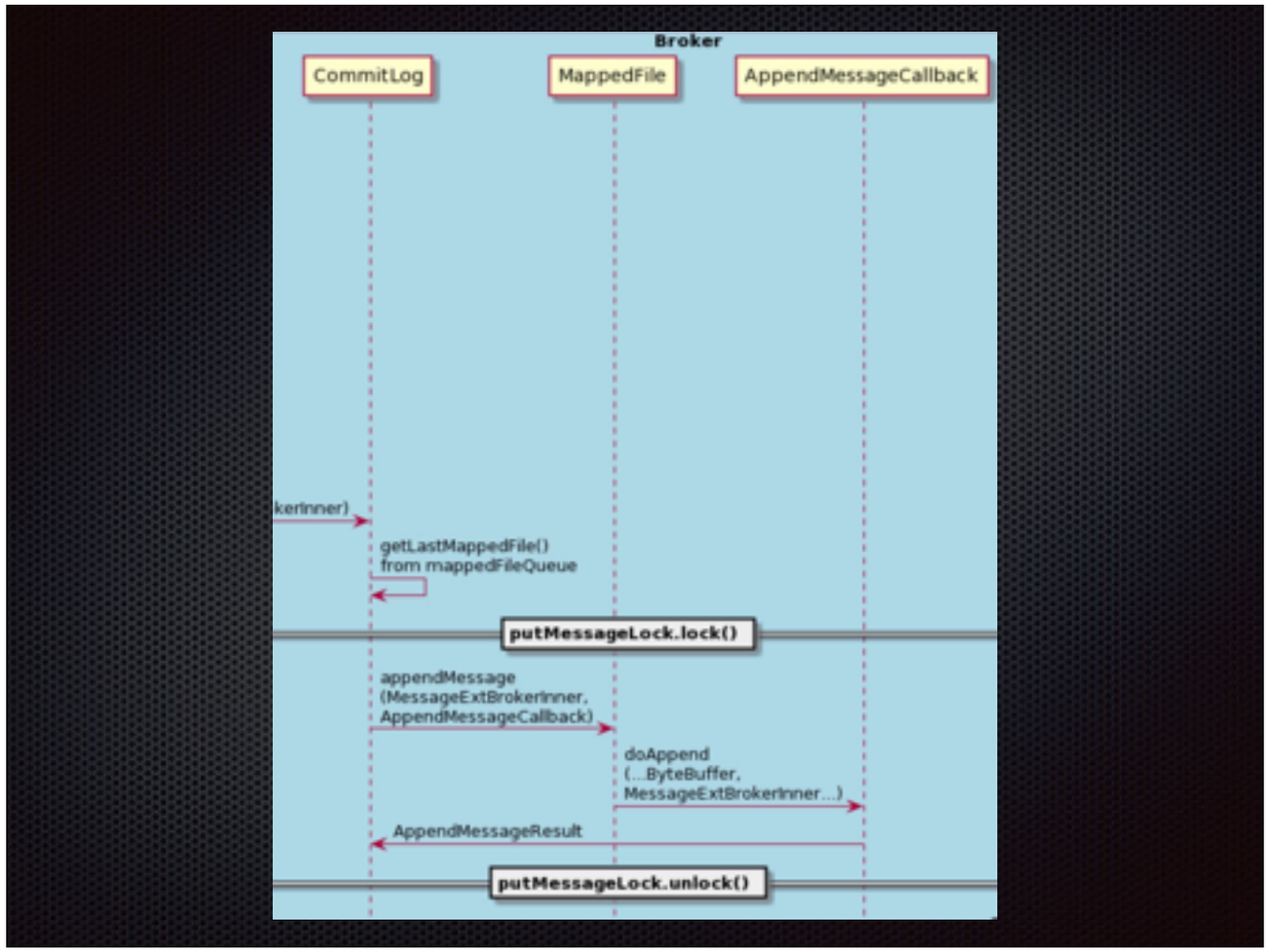
SendMessageProcessor

- 同步阻塞
- 同步非阻塞
- 异步阻塞
- 异步非阻塞

同步阻塞：单线程同步执行，并且等待消息落盘后返回；
同步非阻塞: 单线程同步执行，消息写入缓冲区（byteBuffer）即返回；
异步阻塞：单线程（线程池）异步执行，并且等待消息落盘后返回；
异步非阻塞：单线程（线程池）异步执行，消息写入缓冲区（byteBuffer）即返回；

消息处理时序

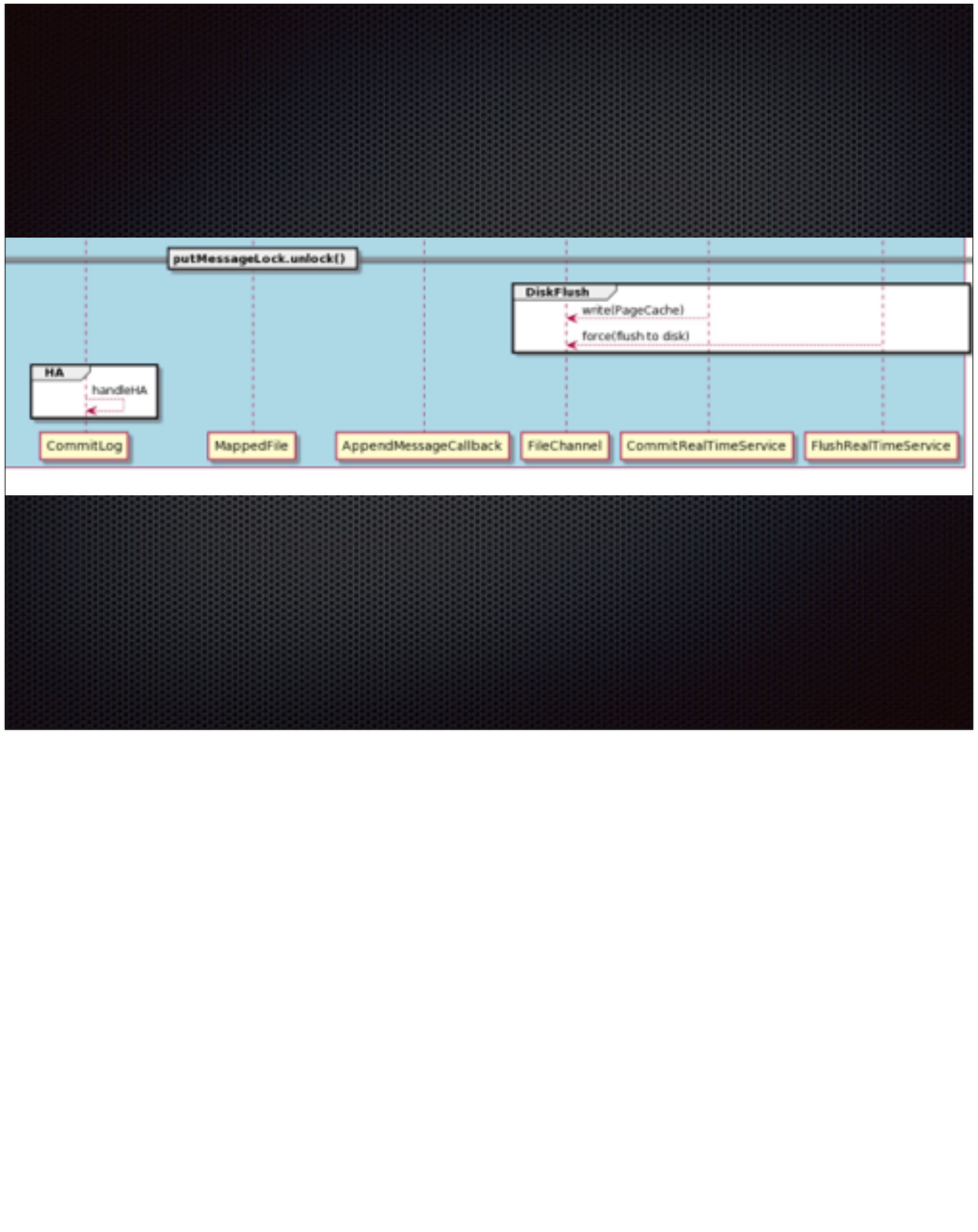

Producer
Client
SendMessageProcessor
DefaultMessageStore
CommitLog
Map
(send message)
processRequest
(RemotingCommand)
asyncProcessRequest
(RemotingCommand)
SendMessageContext
& SendMessageRequestHeader
new MessageExtBrokerInner
with all info
asyncPutMessage
(MessageExtBrokerInner)
putMessage
(MessageExtBrokerInner)
getLastMappedFile()
from mappedFileQueue



使用自旋锁同步对MappedFile的写操作，保证线程安全；

自旋锁实现（自旋锁实现来放置消息，建议在低竞争条件下使用）：

```
public class PutMessageSpinLock implements PutMessageLock {
    //true: Can lock, false : in lock.
    private AtomicBoolean putMessageSpinLock = new AtomicBoolean(true);
    @Override
    public void lock() {
        boolean flag;
        do {
            flag = this.putMessageSpinLock.compareAndSet(true, false);
        }
        while (!flag);
    }
    @Override
    public void unlock() {
        this.putMessageSpinLock.compareAndSet(false, true);
    }
}
```


putMessageLock.unlock()
DiskFlush
write(PageCache)
force(flush to disk)
HA
handleHA
CommitLog
MappedFile
AppendMessageCallback
FileChannel
CommitRealTimeService
FlushRealTimeService

消息提交

- CommitRealTimeService
- 实时将缓存区（byteBuffer）中的数据写入fileChannel，并调用fileChannel.write()方法将数据写入pageCache；

消息落盘

- FlushRealTimeService
- 如果CommitRealTimeService将消息写入pageCache后，会唤醒该线程，该线程完成落盘操作。

Q & A

- topic谁负责创建
- consumergroup的作用
- consumergroup的负载均衡

1. topic在生产者发送消息的时候创建并持久化。
2. 同一个consumergroup中所有的consumer实例必须具有完全相同的主题订阅，否则无法正常消费。因为consumergroup的原因，在消息消费方面实现负载平衡的目标是非常容易的。
3. 对同一组消费者来说，他们组成一个消费者集群，当他们对某个topic进行消费时：

比如，现在有三个队列，三个消费者：

① 获取topic主题下所有的队列（3）

② 获取topic主题下group组下所有的消费者id（3）

③ 排序队列和消费者id（同一个broker的会排在一起，然后按队列id排序），并委托strategy（默任AllocateMessageQueueAveragely）进行队列分配，保证同一个消费组内的消费者分配到的队列是不同的。

④ 将分配给当前消费者的队列传入，进行队列的新增、删除操作

⑤ 处理消费队列分配发生了变化后的逻辑

Thank you!