

JAVA课程项目汇报

学 院： 软件与微电子学院

专 业： 软件工程

组 别： 第十组

成 员： 杨凌锋 杨晓宇 李智 许佳威

任课老师： 褚伟杰

[一、 项目概述 2](#_Toc60517310)

[二、 索引设计 2](#_Toc60517311)

[三、 容器设计 3](#_Toc60517312)

[四、 缓冲区设计 3](#_Toc60517313)

[五、 IO设计 6](#_Toc60517314)

[六、 反思与不足 7](#_Toc60517315)

# 项目概述

此项目为java课程项目大型消息队列以及多线程的应用的汇总报告，在提交的源代码中，一共有6个类，BufferManager，DefaultQueueImpl，DiskManager，Index，IndexItem，QueueStore6个类，接下来的报告中会围绕这6个类的设计与实现分别进行说明，根据功能划分分为索引设计，容器设计，缓冲区设计和IO设计四个方面。

# 索引设计

IndexItem类作为我们的索引类，拥有三个成员变量offset，startMsgIndex，msgNum，offset代表此索引指向的文件块在文件中存储的位置，startMsgIndex代表此索引指向的文件块中存储的第一条消息的编号或者说消息在队列中的顺序编号，msgNum代表该索引指向的文件块中消息的数量。

获取从队列某个位置开始的指定数量消息中的数据块的索引数组的方法由Index. getBufIndexs(long offset, long num)实现，由getFirst标识来表示当前索引是不是需要读取的索引的第一个，顺序判断内存中的索引是不是需要读取的第一个，判断方法是offset是否被某个索引指向的数据块的开始和结束位置夹在中间，读取结束的标志是几个索引的消息数量之和大于或等于num。

class IndexItem {  
 public long offset;//文件中的存储位置  
 public int startMsgIndex;//开始消息在某个队列中的编号  
 public int msgNum;//消息数量  
  
 public IndexItem(long offset, int startMsgIndex, int msgNum) {  
 this.offset = offset;  
 this.startMsgIndex = startMsgIndex;  
 this.msgNum = msgNum;  
 }  
}

# 容器设计

DefaultImpl中如下所示的两个变量完成了容器设计的功能，其中Empty用于表示读出的信息为空的情况，利用java自带的同步concurrentHashMap存储最为重要的不同队列的缓冲区，每个缓冲区由队列名标明，并为每一个缓冲区配备一个缓冲区管理类BufferManager，对于磁盘的写入写出会在BufferManager和DiskManager中进行阐述。

public static Collection<byte[]> *EMPTY* = new ArrayList<>();  
private ConcurrentHashMap<String, BufferManager> bufferManagers;

值得一提的是在文件通道的使用中，使用RandomAccess类获取读取流后，写入时因为屏蔽读操作且屏蔽其他写操作，所以只需使用RandomAccess.getChannel()方法获取可用的通道即可，但是在读取时，需要根据当前线程判断是否需要添加新的文件通道进行文件的读写。这一部分在IO设计中进行详细的阐述。

# 缓冲区设计

整个项目中任务量最大的部分在缓冲区设计部分，经过测试，缓冲区的大小设置为8KB能够具有一个比较不错的性能，使用AtomicLong类的一个变量来记录当前队列中的数据块的数量，并使用AtomicLong类自带的方法getAndIncrement()来实现数据块数量的同步增长，利用java.nio.ByteBuffer类来作为为某个队列分配的缓冲区，而且由于频繁的本地IO所以使用ByteBuffer.allocateDirect()方法来进行缓冲区的初始化。除此之外，buffermanager中还设置了两个变量来存储整个队列中的消息数量和当前缓冲区中的消息数量。

private int currentMsgNum = 0;//整个队列中的消息数量  
private int bufferMagNum = 0;//当前缓冲区中的消息数量

BufferManager类共有put()和get()两个主要函数，skipNMsgInBuf(),getNextMsgInBuf()，writeToDisk()三个辅助函数，put()和get()的执行流程图如图1，图2所示(添加从DefaultQueueImpl的put()和get()方法的执行流程)：

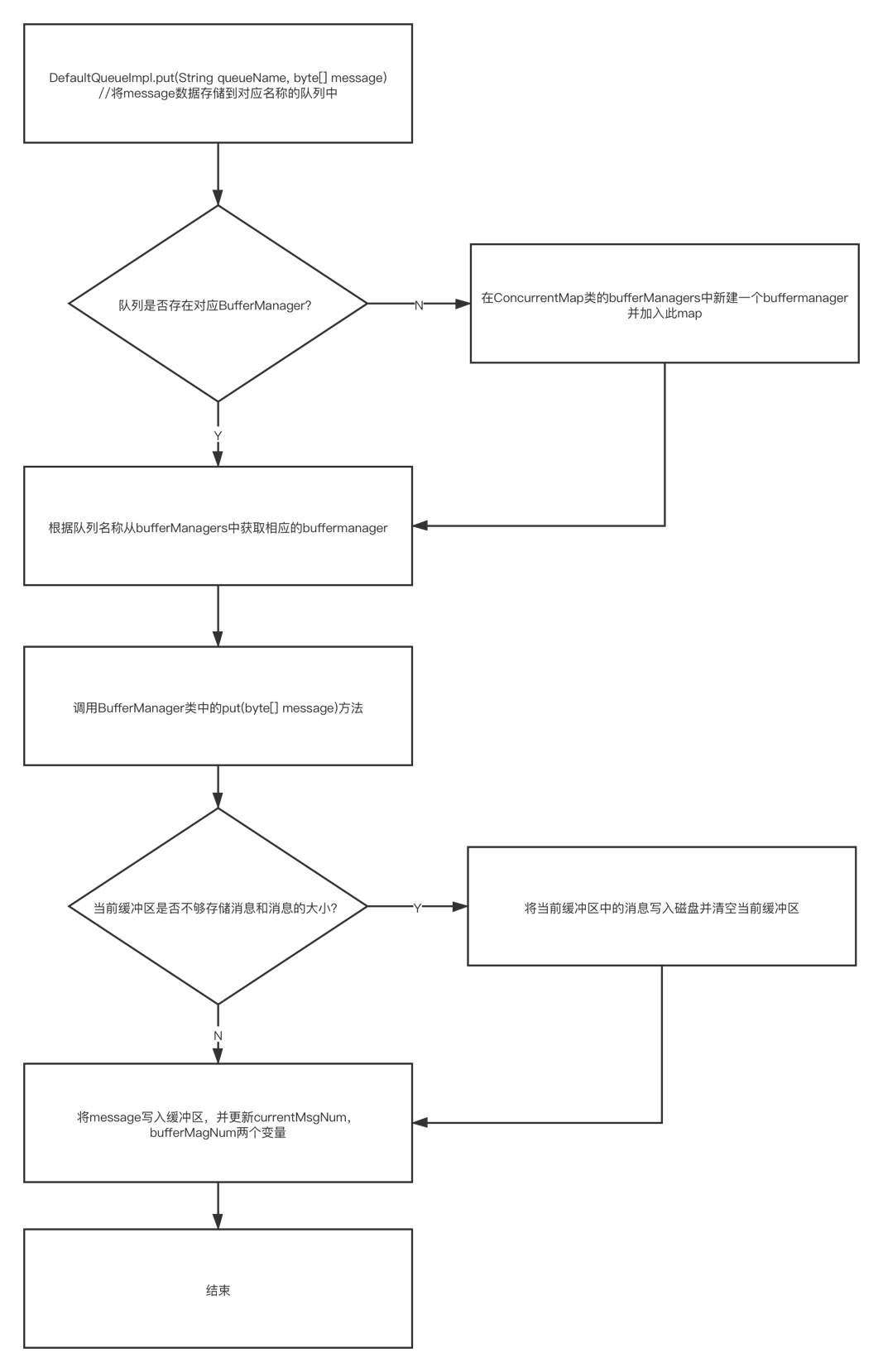


图1

图示

描述已自动生成

图2

图1和图2展示了BufferManager类的核心功能get()和put(),关于三个辅助函数分别作出说明：

Void skipNMsgInBuf(int n)：改变当前缓冲区的指针指向位置，改为跳过当前指向位置跳过n个消息后的位置，这样的原因是某个索引块指向的数据块被offset划分为两部分，并不是所有的消息都要被读取，所以跳过不需要被读取的部分。

byte[] getNextMsgInBuf()：从消息缓冲区中获取当前缓冲区指针指向的下个消息，重复操作获取的消息集合作为get()方法的返回值。

Void writeToDisk()：将当前缓冲区的内容写入磁盘，先将缓冲区的position指针指向0，再将当前bufferManager的bufIndex变量同步自增1，将索引添加到当前bufferManager的索引数组中，索引在磁盘中的位置即为bufIndex变量\*每个缓冲区的大小（即8K），索引中首个消息的编号为currentMsgNum-bufferMagNum，索引中消息的数量为bufferMagNum。

writeToDisk（）函数中的Index.add(long offset, int startMsg, int msgNum)方法和DiskManager.writeToFile(ByteBuffer buffer, long position)方法分别在IO设计和索引设计中阐述。

# IO设计

IO设计和DiskManager类和BufferManager有关，这里涉及到的类有java.io.RandomAccessFile和java.nio.channels.FileChannel以及java.util.concurrent.ConcurrentHashMap三个类，RandomAccessFile是一个高性能读写流，相当于datainputStream和dataoutputStream用于内存和磁盘的交互，FileChannel是nio包中提供的一个类，提供一个通道为文件进行同步读写，FileChannel运行在阻塞模式下，且无法设置为非阻塞模式。

DiskManager类初始化一个文件读写流后，再新建一个concurrentHashMap用于记录不同的线程和文件通道之间的映射，并用于实现以下两个函数：

writeToFile(ByteBuffer buffer, long position)：用于将buffer中的内容写到文件的position位置中去，position由写入之前BufferManager中的curBufIndex\*8K来确定，curBufIndex代表当前队列有多少个索引，写入的方式为使用randomAccessFile.getChannel()获取写入的通道，再调用通道的写入方法channel.write(buffer, position)将内容写入即可。

readAbBuf(ByteBuffer buffer, long position)：position标明了某个索引指向的数据块在文件中存储的位置，readFileChannels.get(Thread)用于通过线程的映射获取相应的文件通道，若通道存在则直接获取，若不存在，则新建一个并加入到此类的concurrentHashMap映射中去，并通过通道将postion中的一个buffer大小的数据块读入缓冲区中。

整个DiskManager的执行流程如图3所示：

图示

描述已自动生成

图3

# 反思与不足

1. 在索引数组的添加时采取顺序遍历的方式，未采用二分查找的方式或者其他方式，二分查找的情况下效率可能会有提升。
2. 将某个队列索引存储在一个list中，存在于内存中，可以减少内存使用的方法是将list改为数组。
3. 在缓冲区的性能测试中只采取了8K，4K和2K进行测试，而测试也未进行多次取平均，恐与服务器当前的负载情况有关，所以8K是否是一个合理的缓冲区大小也有待商榷。