How to avoid OutOfMemoryError when using Bytebuffers and NIO?

Asked 16 years, 3 months ago Modified 6 years ago Viewed 10k times

3

I'm using `ByteBuffers` and `FileChannels` to write binary data to a file. When doing that for big files or successively for multiple files, I get an `OutOfMemoryError` exception. I've read elsewhere that using `Bytebuffers` with NIO is broken and should be avoided. Does any of you already faced this kind of problem and found a solution to efficiently save large amounts of binary data in a file in java?

Is the jvm option `-XX:MaxDirectMemorySize` the way to go?

java nio bytebuffer filechannel

Share

Improve this question

Follow

edited Feb 22, 2014 at 21:38

Durandal

5,663 ● 5 ● 37 ● 50

asked Aug 26, 2008 at 17:23

jumar

5,390 ● 8 ● 47 ● 42

6 Answers

Sorted by: Highest score (default)

6

✓

I would say don't create a huge ByteBuffer that contains ALL of the data at once. Create a much smaller ByteBuffer, fill it with data, then write this data to the FileChannel. Then reset the ByteBuffer and continue until all the data is written.

Share Improve this answer Follow

answered Aug 26, 2008 at 17:26

Tim Frey
9,931 ●9 ●45 ●61

6

Check out Java's **[Mapped Byte Buffers]**, also known as 'direct buffers'. Basically, this mechanism uses the OS's virtual memory paging system to 'map' your buffer directly to disk. The OS will manage moving the bytes to/from disk and memory auto-magically, very quickly, and you won't have to worry about changing your virtual machine options. This will also allow you to take advantage of NIO's improved performance over traditional java stream-based i/o, without any weird hacks.

The only two catches that I can think of are:

1. On 32-bit system, you are limited to just under 4GB *total for all mapped byte buffers*. (That is actually a limit for my application, and I now run on 64-bit architectures.)

2. Implementation is JVM specific and not a requirement. I use Sun's JVM and there are no problems, but YMMV.

Kirk Pepperdine (a somewhat famous Java performance guru) is involved with a website, www.JavaPerformanceTuning.com, that has some more MBB details: **NIO Performance Tips**

Share Improve this answer Follow

edited Aug 26, 2008 at 18:47

answered Aug 26, 2008 at 18:01

Stu Thompson
38.9k ● 19 ● 111 ● 156

Thanks for pointing out that there is a limit for all mapped byte buffers (of just my application or all on the OS??) In my case, I get the stupid OutOfMemoryException even when I try one MappedByteBuffer for a file of around 1.6GB! But why? How am I supposed to find out just how large my remaining space is?? Help! – Zordid Sep 21, 2012 at 13:58

@Zordid Ummm... `MappedByteBuffer` s (Mapped is key!) generally do not cause `OutOfMemoryException` s. Something else is broken. I would suggest creating a new question here on StackOverflow...with code! Someone is likely to be able to help you out. – Stu Thompson Sep 21, 2012 at 16:21

If you access files in a *random fashion* (read here, skip, write there, move back) then you have a problem ;-)

1

But if you only write big files, you should **seriously** consider using streams. `java.io.FileOutputStream` can be used directly to write file byte after byte or wrapped in any other stream (i.e. `DataOutputStream`, `ObjectOutputStream`) for convenience of writing floats, ints, Strings or even serializeable objects. Similar classes exist for reading files.

Streams offer you convenience of manipulating **arbitrarily large files in (almost) arbitrarily small memory**. They are preferred way of accessing file system in vast majority of cases.

Share Improve this answer

Follow

edited Dec 20, 2018 at 12:52

Akash Thakare

23k ● 11 ● 63 ● 87

answered Aug 26, 2008 at 17:35

Marcin

7,994 ● 7 ● 46 ● 51

0

Using the transferFrom method should help with this, assuming you write to the channel incrementally and not all at once as previous answers also point out.

Share Improve this answer

Follow

answered Aug 26, 2008 at 18:51

Cagatay

1,372 ● 1 ● 12 ● 16

0

This can depend on the particular JDK vendor and version.

There is a bug in GC in some Sun JVMs. Shortages of direct memory will not trigger a GC in the main heap, but the direct memory is pinned down by garbage direct ByteBuffers in the main heap. If the main heap is mostly empty they many not be collected for a long time.

This can burn you even if you aren't using direct buffers on your own, because the JVM may be creating direct buffers on your behalf. For instance, writing a non-direct ByteBuffer to a SocketChannel creates a direct buffer under the covers to use for the actual I/O operation.

The workaround is to use a small number of direct buffers yourself, and keep them around for reuse.

Share Improve this answer Follow

answered Sep 26, 2008 at 15:14

Darron
21.6k ● 5 ● 51 ● 54

0

The previous two responses seem pretty reasonable. As for whether the command line switch will work, it depends how quickly your memory usage hits the limit. If you don't have enough ram and virtual memory available to at least triple the memory available, then you will need to use one of the alternate suggestions given.

Share Improve this answer

edited Dec 20, 2018 at 12:53

Follow

Akash Thakare
23k ● 11 ● 63 ● 87

answered Aug 26, 2008 at 18:02

Dana the Sane
15.2k ● 8 ● 60 ● 81