How to unmap a file from memory mapped using FileChannel in java?

I am mapping a file("sample.txt") to memory using `FileChannel.map()` and then closing the channel using `fc.close()`. After this when I write to the file using FileOutputStream, I am getting the following error:

> java.io.FileNotFoundException: sample.txt (The requested operation cannot be per formed on a file with a user-mapped section open)

```
File f = new File("sample.txt");
RandomAccessFile raf = new RandomAccessFile(f,"rw");
FileChannel fc = raf.getChannel();
MappedByteBuffer mbf = fc.map(FileChannel.MapMode.READ_ONLY, 0, fc.size());
fc.close();
raf.close();

FileOutputStream fos = new FileOutputStream(f);
fos.write(str.getBytes());
fos.close();
```

I presume this may be due to file being still mapped to the memory even after I close the `FileChannel`. Am I right?. If so, how can I "unmap" the file from memory?(I can't find any methods for this in the API). Thanks.

Edit: Looks like it(adding an unmap method) was submitted as RFE to sun some time back: http://bugs.sun.com/view_bug.do?bug_id=4724038

java memory-mapped-files

edited Jun 4 '10 at 10:52

asked Jun 4 '10 at 9:47
learner135
133 1 2 6

1 I hit the same problem painfully. In my case I was trying to map thousands of files into the memory and it results `OutOfMemoryError`, which I believe is the result of exhausted file handlers. Having no instrument to unmap leads to unpredictive behaviour, even though map will try to `sleep(100)` & `System.gc()` on `OutOfMemoryError` – it just postpones the convulsions. – dma_k May 16 '12 at 14:35

8 Answers

From the `MappedByteBuffer` javadoc:

> A mapped byte buffer and the file mapping that it represents remain valid until the buffer itself is garbage-collected.

Try calling `System.gc()`? Even that's only a suggestion to the VM.

answered Jun 4 '10 at 9:58
scompt.com
15.9k 6 58 108

Thanks a lot. I got it working but won't relying on System.gc() yield unexpected results on different runs? – learner135 Jun 4 '10 at 10:10

Yup. If that solved your problem, then you at least know what the problem is. Now you can decide if you want to do something crazy like loop calling `System.gc()` or reconsider your implementation. – scompt.com Jun 4 '10 at 10:12

I tried System.gc() after removing all references, but I still got the error. – Tim Cooper Jun 29 '11 at 16:02

1 Won't work if `-XX:+DisableExplicitGC` is set – Bass Aug 5 '15 at 8:58

Following static method could be used:

```
public static void unmap(MappedByteBuffer buffer)
{
   sun.misc.Cleaner cleaner = ((DirectBuffer) buffer).cleaner();
   cleaner.clean();
}
```

But this is unsafe solution because of following: 1) Lead to failures if someone use MappedByteBuffer after unmap 2) It relies on MappedByteBuffer implementation details

answered Feb 17 '11 at 23:25
Timur Yusupov
479 6 9

1 This "solution" should not be used. It is both unportable and unsafe. – kittylyst Aug 5 '13 at 14:49

13 @kittylyst problem should not be solved. Company should not sell its products. Developer should not be paid. – stepancheg Aug 16 '13 at 15:25

@stephancheg See my answer below for how to actually (mostly) solve this problem. – kittylyst Aug 17 '13 at 12:52

2 The danger is that you'll crash the JVM if the buffer is accessed again. See the comments: bugs.sun.com/view_bug.do?bug_id=4724038 – Aleksandr Dubinsky Dec 19 '13 at 11:57

The solution is "portable to some extent", as `sun.misc.Cleaner` is present in all major JDK flavours (*Oracle JDK*, *OpenJDK* incl. *Zulu*, IBM *J9*), except for *GCJ* and *Apache Harmony*. It is also likely to be preserved in JDK 1.9 (see openjdk.java.net/jeps/260) – Bass Aug 5 '15 at 9:04

|

[WinXP,SunJDK1.6] I had a mapped ByteBuffer taken from filechannel. After reading SO posts finally manag
cleaner through reflection without any sun.* package imports. No longer file lock is lingering.

```
FileInputStream fis = new FileInputStream(file);
FileChannel fc = fis.getChannel();
ByteBuffer cb = null;
try {
    long size = fc.size();
    cb = fc.map(FileChannel.MapMode.READ_ONLY, 0, size);
    ...do the magic...
finally {
    try { fc.close(); } catch (Exception ex) { }
    try { fis.close(); } catch (Exception ex) { }
    closeDirectBuffer(cb);
}

private void closeDirectBuffer(ByteBuffer cb) {
    if (!cb.isDirect() return;

    // we could use this type cast and call functions without reflection code,
    // but static import from sun.* package is risky for non-SUN virtual machine.
    //try { ((sun.nio.ch.DirectBuffer)cb).cleaner().clean(); } catch (Exception ex) { }
    try {
        Method cleaner = cb.getClass().getMethod("cleaner");
        cleaner.setAccessible(true);
        Method clean = Class.forName("sun.misc.Cleaner").getMethod("clean");
        clean.setAccessible(true);
        clean.invoke(cleaner.invoke(cb));
    } catch(Exception ex) { }
    cb = null;
}
```

Ideas were taken from these posts.
* How to unmap a file from memory mapped using FileChannel in java?
* Examples of forcing freeing of native memory direct ByteBuffer has allocated, using sun.misc.Unsafe?
*
https://github.com/elasticsearch/elasticsearch/blob/master/src/main/java/org/apache/lucene/store/bytebuffer/
cator.java#L40

edited Oct 18 '13 at 10:53

Works for us (JDK 7 64 bits, Red Hat). It avoids having compilation warnings on "sun" package (we always use the same Oracle VM in a controlled and per-defined environment) – xav Jul 10 '14 at 13:20

To work around this bug in Java, I had to do the following, which will work ok for small to medium-sized files:

```
    // first open the file for random access
    RandomAccessFile raf = new RandomAccessFile(file, "r");

    // extract a file channel
    FileChannel channel = raf.getChannel();

    // you can memory-map a byte-buffer, but it keeps the file locked
    //ByteBuffer buf =
    //        channel.map(FileChannel.MapMode.READ_ONLY, 0, channel.size());

    // or, since map locks the file... just read the whole file into memory
    ByteBuffer buf = ByteBuffer.allocate((int)file.length());
    int read = channel.read(buf);

    // .... do something with buf

    channel.force(false);  // doesn't help
    channel.close();       // doesn't help
    channel = null;        // doesn't help
    buf = null;            // doesn't help
    raf.close();           // try to make sure that this thing is closed!!!!!
```

answered Jan 19 '12 at 9:22
Mr Ed
3,082 1 9 10

sun.misc.Cleaner javadoc says:

> General-purpose phantom-reference-based cleaners. Cleaners are a lightweight and more robust alternative to finalization. They are lightweight because they are not created by the VM and thus do not require a JNI upcall to be created, and because their cleanup code is invoked directly by the reference-handler thread rather than by the finalizer thread. They are more robust because they use phantom references, the weakest type of reference object, thereby avoiding the nasty ordering problems inherent to finalization. A cleaner tracks a referent object and encapsulates a thunk of arbitrary cleanup code. Some time after the GC detects that a cleaner's referent has become phantom-reachable, the reference-handler thread will run the cleaner. Cleaners may also be invoked directly; they are thread safe and ensure that they run their thunks at most once. Cleaners are not a replacement for finalization. They should be used only when the cleanup code is extremely simple and straightforward. Nontrivial cleaners are inadvisable since they risk blocking the reference-handler thread and delaying further cleanup and finalization.

Running System.gc() is acceptable solution if your buffers total size is small, but if I was mapping gigabytes of files I would try to implement like this:

```
((DirectBuffer) buffer).cleaner().clean()
```

But! Make sure you don't access that buffer after cleaning or you will end up with:

> A fatal error has been detected by the Java Runtime Environment: EXCEPTION_ACCESS_VIOLATION (0xc0000005) at pc=0x0000000002bcf700, pid=7592, tid=10184 JRE version: Java(TM) SE Runtime Environment (8.0_40-b25) (build 1.8.0_40-b25) Java VM: Java HotSpot(TM) 64-Bit Server VM (25.40-b25 mixed mode windows-amd64 compressed oops) Problematic frame: J 85 C2 java.nio.DirectByteBuffer.get(I)B (16 bytes) @ 0x0000000002bcf700 [0x0000000002bcf6c0+0x40] Failed to write core dump. Minidumps are not enabled by default on client versions of Windows An error report file with more information is saved as: C:\Users\?????\Programs\testApp\hs_err_pid7592.log Compiled method (c2) 42392 85 4 java.nio.DirectByteBuffer::get (16 bytes) total in heap
> [0x0000000002bcf590,0x0000000002bcf828] = 664 relocation
> [0x0000000002bcf6b0,0x0000000002bcf6c0] = 16 main code
> [0x0000000002bcf6c0,0x0000000002bcf760] = 160 stub code
> [0x0000000002bcf760,0x0000000002bcf778] = 24 oops
> [0x0000000002bcf778,0x0000000002bcf780] = 8 metadata
> [0x0000000002bcf780,0x0000000002bcf798] = 24 scopes data
> [0x0000000002bcf798,0x0000000002bcf7e0] = 72 scopes pcs
> [0x0000000002bcf7e0,0x0000000002bcf820] = 64 dependencies
> [0x0000000002bcf820,0x0000000002bcf828] = 8

Good luck!

answered Jul 23 '15 at 16:21
Dmitrius
51 2

The mapped memory is used until it is freed by the garbage collector.

From FileChannel docs

> A mapping, once established, is not dependent upon the file channel that was used to create it. Closing the channel, in particular, has no effect upon the validity of the mapping.

From MappedByteBuffer java doc

> A mapped byte buffer and the file mapping that it represents remain valid until the buffer itself is garbage-collected.

So I would suggest ensuring there are no remaining references to the mapped byte buffer and then requesting a garbage collection.

answered Jun 4 '10 at 9:59
BenM
2,875 16 24

1 There is no way to force a GC. Not even through `System.gc()` . – aioobe Jun 4 '10 at 10:08

1 that's why I said request – BenM Jun 4 '10 at 10:55

Just a comment about a statement made here "There is no way to force a GC. Not even through System.gc()" Actually there is. If you make sure all the application threads are idle or sleeping, System.gc() will run the GC every time, at least on Oracle's VM. – user1353729 Jul 25 '12 at 21:51

The only way to force a GC is to provoke an `OutOfMemoryError` . The specs guarantee that before the exception is thrown, the GC was run. – Bombe Jun 26 '14 at 18:32

If the mapped file buffer object can be guaranteed to be eligible for garbage collection, you don't need to GC the whole VM to get the buffer's mapped memory to be released. You can call System.runFinalization() . This will call the finalize() method on the mapped file buffer object (if it there are no references to it in your app threads) which will release the mapped memory.

answered Aug 1 '13 at 11:44
somedude
1

1 No. Use try-with-resources instead. Finalization should *never* be used. – kittylyst Aug 5 '13 at 14:53

Per the docs, finalization is the only way that unmapping occurs. Other than undocumented methods relying on sun.* classes, this is the only actual solution to the problem. Try-with-resources just closes the channel, which the OP already did anyway. – Jules Dec 18 '13 at 23:48

Doesn't the GC need to be run to determine which objects are eligible for finalization? Maybe better to call the GC and then `runFinalization` ? – Aleksandr Dubinsky Dec 19 '13 at 11:41

The correct solution here is to use try-with-resources.

This allows the creation of the Channel & the other resources to be scoped to a block. Once the block exits, the Channel & other resources are gone & subsequently cannot be used (as nothing has a reference to them).

The memory-mapping still won't be undone until the next GC run, but at least there aren't any dangling references to it.

answered Aug 5 '13 at 14:56
kittylyst
3,102 11 29

This does not solve anything. Problem is not with dangling references, but with mapping itself which remains alive. – stepancheg Aug 17 '13 at 18:06

1 The problem is that the memory mapping causes the file to be locked at the OS level, meaning that it is impossible to write to it using non-memory-mapped operations or to delete it. This supposed solution does nothing to solve this problem. – Jules Dec 18 '13 at 23:46