

HowToDoInJava

Python

Java

Spring Boot



Related Tutorials

[Home](#) / [Java](#) / [Java New Input/Output](#) / [Java MappedByteBuffer](#)

Java MappedByteBuffer

Last Modified: December 26, 2020

Learn about **Java memory-mapped files** and learn to read and write **content from a memory mapped file** with the help of `RandomAccessFile` and `MemoryMappedBuffer`.

1. Java Memory-mapped IO

If you know [how java IO works at lower level](#), then you will be aware of buffer handling, memory paging and other such concepts. For conventional file I/O, in which user processes issue `read()` and `write()` system calls to transfer data, there is almost always one or more copy operations to move the data between these filesystem pages in kernel space and a memory area in user space. This is because there is not usually a one-to-one alignment between filesystem pages and user buffers.

There is, however, a special type of I/O operation supported by most operating systems that allows user processes to take maximum advantage of the **page-oriented nature of system I/O** and **completely avoid buffer copies**. This is called **memory-mapped I/O** and we are going to learn few things here around **memory-mapped files**.

2. Java Memory-Mapped Files

Memory-mapped I/O uses the filesystem to establish a virtual memory mapping from user space directly to the applicable filesystem pages. With a memory-mapped file, we can **pretend** that the entire file is in memory and that we can access it by simply treating it as a very large array. This approach greatly simplifies the code we write in order to modify the file.

Read More : [Working With Buffers](#)

To do both writing and reading in memory mapped files, we start with a `RandomAccessFile`, get a channel for that file. **Memory mapped byte buffers are created via the `FileChannel.map()` method**. This class extends the `ByteBuffer` class with operations that are specific to memory-mapped file regions.

A mapped byte buffer and the file mapping that it represents remain valid until the buffer itself is garbage-collected. Note that you **must specify the starting point and the length of the region that you want to map in the file**; this means that you have the **option to map smaller regions of a large file**.

Example 1: Writing to a memory mapped file

如果您知道java

IO在较低级别的工作方式，那么您就会了解缓冲区处理、内存分页以及其他类似的概念。对于传统的文件I/O，其中用户进程发出`read()`和`write()`传输数据的系统调用，几乎总是有一个或多个复制操作在数据之间移动这些文件系统页位于内核空间，内存区域位于用户空间。这是因为平时没有文件系统页和用户缓冲区之间的一对一对齐。

然而，有一种特殊类型的I/O操作被大多数操作系统支持，允许用户进程最大限度地利用系统I/O面向页面的特性，并完全避免缓冲区拷贝。这被称为内存映射I/O，我们将在这里学习一些东西内存映射文件。

内存映射I/O使用文件系统建立直接从用户空间到虚拟内存映射适用的文件系统页面。对于内存映射文件，我们可以假设整个文件都在内存中，我们可以把它当作一个很大的数组来访问它。这种方法大大简化了为了修改文件而编写的代码。

MemoryMappedFileExample.java

```
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class MemoryMappedFileExample
{
    static int length = 0x8FFFFFFF;

    public static void main(String[] args) throws Exception
    {
        try(RandomAccessFile file = new RandomAccessFile("howtodoinjava.dat", "rw"))
        {
            MappedByteBuffer out = file.getChannel()
                .map(FileChannel.MapMode.READ_WRITE, 0, length);

            for (int i = 0; i < length; i++)
            {
                out.put((byte) 'x');
            }

            System.out.println("Finished writing");
        }
    }
}
```

后面这两个参数指定映射的位置和长度

The file created with the above program is 128 MB long, which is probably larger than the space your OS will allow. The file appears to be accessible all at once because only portions of it are brought into memory, and other parts are swapped out. This way a very large file (up to 2 GB) can easily be modified.

3. File Mapping Modes

Like conventional file handles, file mappings can be writable or read-only.

您会注意到没有unmap()方法。一旦建立，映射将保持有效，直到MappedByteBuffer对象被垃圾回收。

- The first two mapping modes, **MapMode.READ_ONLY** and **MapMode.READ_WRITE**, are fairly obvious. They indicate whether you want the mapping to be read-only or to allow modification of the mapped file.
- The third mode, **MapMode.PRIVATE**, indicates that you want a copy-on-write mapping. This means that any modifications you make via **put()** will result in a private copy of the data that only the **MappedByteBuffer** instance can see. No changes will be made to the underlying file, and any changes made will be lost when the buffer is garbage collected. Even though a copy-on-write mapping prevents any changes to the underlying file, you must have opened the file for read/write to set up a **MapMode.PRIVATE** mapping. This is necessary for the returned **MappedByteBuffer** object to allow **put()**s.

You'll notice that there is no **unmap()** method. Once established, a mapping remains in effect until the **MappedByteBuffer** object is garbage collected.

Also, mapped buffers are not tied to the channel that created them. Closing the associated **FileChannel** does not destroy the mapping; only disposal of the buffer object itself breaks the mapping.

A **MemoryMappedBuffer** has a fixed size, but the file it's mapped to is elastic. Specifically, if a file's size changes while the mapping is in effect, some or all of the buffer may become inaccessible, undefined data could be returned, or unchecked exceptions could be thrown.

Be careful about how files are manipulated by other threads or external processes when they are memory-mapped.

另外，映射的缓冲区不会绑定到创建它们的通道。关闭关联的FileChannel不破坏映射；只有缓冲区对象本身的处理才会破坏映射。

MemoryMappedBuffer的大小是固定的，但是它映射到的文件是有弹性的。具体来说，如果一个文件当映射生效时，大小发生变化，部分或全部缓冲区可能变得不可访问，可能会返回未定义的数据，或者引发未检查的异常。当其他线程或外部进程对文件进行操作时，要小心内存映射。

4. Benefits of Memory Mapped Files

Memory-Mapped IO have several advantages over normal I/O:

1. The user process sees the file data as memory, so there is no need to issue `read()` or `write()` system calls.
用户进程将文件数据视为内存，因此不需要发出read()或write()系统调用。
2. As the user process touches the mapped memory space, page faults will be generated automatically to bring in the file data from disk. If the user modifies the mapped memory space, the affected page is automatically marked as dirty and will be subsequently flushed to disk to update the file.
3. The virtual memory subsystem of the operating system will perform intelligent caching of the pages, automatically managing memory according to system load.
4. The data is always page-aligned, and no buffer copying is ever needed.
5. Very large files can be mapped without consuming large amounts of memory to copy the data.

5. How to read a Memory-Mapped File

To read a file using memory mapped IO, use below code template:

```
import java.io.File;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class MemoryMappedFileReadExample
{
    private static String bigExcelFile = "bigFile.xls";

    public static void main(String[] args) throws Exception
    {
        try (RandomAccessFile file = new RandomAccessFile(new File(bigExcelFile), "r"))
        {
            //Get file channel in read-only mode
            FileChannel fileChannel = file.getChannel();

```

数据总是页面对齐的，不需要缓存复制。

当用户进程接触映射的内存空间时，将自动生成页面错误从磁盘导入文件数据。如果用户修改了映射的内存空间，则影响页面为自动标记为dirty，并将随后刷新到磁盘以更新文件。

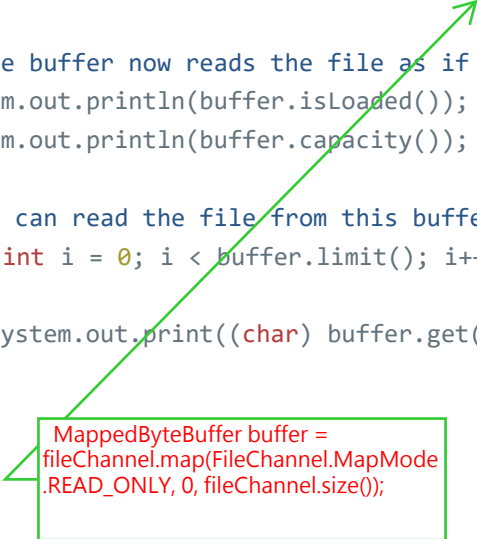
操作系统的虚拟内存子系统将执行页面的智能缓存，根据系统负载自动管理内存。

可以在不消耗大量内存的情况下映射非常大的文件来复制数据。

```
//Get direct byte buffer access using channel.map() operation
MappedByteBuffer buffer = fileChannel.map(FileChannel.MapMode.READ_ONLY, 0, fileCh

// the buffer now reads the file as if it were loaded in memory.
System.out.println(buffer.isLoaded()); //prints false
System.out.println(buffer.capacity()); //Get the size based on content size of fi

//You can read the file from this buffer the way you like.
for (int i = 0; i < buffer.limit(); i++)
{
    System.out.print((char) buffer.get()); //Print the content of file
}
}
}
}
```



MappedByteBuffer buffer =
fileChannel.map(FileChannel.MapMode
.READ_ONLY, 0, fileChannel.size());

6. How to write into a Memory-Mapped File

To write data into a file using memory mapped IO, use below code template:

```
import java.io.File;
import java.io.RandomAccessFile;
import java.nio.MappedByteBuffer;
import java.nio.channels.FileChannel;

public class MemoryMappedFileWriteExample {
    private static String bigTextFile = "test.txt";

    public static void main(String[] args) throws Exception
    {
        // Create file object
        File file = new File(bigTextFile);

        //Delete the file; we will create a new file
        file.delete();

        try (RandomAccessFile randomAccessFile = new RandomAccessFile(file, "rw"))
        {
            // Get file channel in read-write mode
            FileChannel fileChannel = randomAccessFile.getChannel();

            // Get direct byte buffer access using channel.map() operation
            MappedByteBuffer buffer = fileChannel.map(FileChannel.MapMode.READ_WRITE, 0, 4096);

            //Write the content using put methods
            buffer.put("howtodoinjava.com".getBytes());
        }
    }
}
```

Drop me your comments and thoughts in the comments section.

Happy Learning !!

~ ADVERTISEMENT ~

Share this:



Subscribe to Blog

Enter your email address to subscribe and receive notifications of new posts by email.
Join 3,810 other subscribers

~ ADVERTISEMENT ~

About Lokesh Gupta

A family guy with fun loving nature. Love computers, programming and solving everyday problems.
Find me on [Facebook](#) and [Twitter](#).

Feedback, Discussion and Comments

LH

November 28, 2018

You really should update your code to show how to close the `RandomAccessFile` and `FileChannel` (e.g. in try-with-resources), and set the `MappedByteBuffer` reference to null (to encourage GC), once you have finished with the file.

[Reply](#)

Hossein

April 27, 2018

Hi,

Is there any way to access memory files like buffers , streams or ... with an address ?
in an android project ,
for example I want to access to these memory objects like access to any file in phone storage with
an its address

[Reply](#)

Bao

December 8, 2017

0x8FFFFFFF is not 128 MB

0x80000000 is 128 MB

[Reply](#)

dapinder

May 18, 2017

i am fetching data of around 15GB from a Database and creating a file from that. Will memory mapped file help me ?

[Reply](#)

Yuest

October 1, 2016

Must mention that releasing the memory does not work on some unices as expected. Then you need explicitly call sun package to release memory. Off course the warning for usin sun internal api eill show up.

[Reply](#)

hamza

April 14, 2016

to use mapped memory file i need to know just the file name or all the path ??!!!!

[Reply](#)

iqbal

May 25, 2016

have you heard about relative paths?

[Reply](#)

Sudhagar

September 9, 2015

Hi,

I like to check how to read a file more that 2GB. Can I define the offset and split the file and make more than one read? I was not successful doing it. Can you please throw your ideas on it.

Thanks

Sudhagar C

[Reply](#)[Lokesh Gupta](#)

September 10, 2015

Have you tried code written in "Reading a Memory-Mapped File"? How big file it was able to handle in your system?

[Reply](#)[MikaelJ](#)

May 9, 2015

I'd like to add that I've created a library, MappedBus <https://github.com/caplogic/mappedbus>), which supports having multiple processes write records in order to the same memory mapped file as well as having multiple processes reading records from it.

[Reply](#)[Lokesh Gupta](#)

May 10, 2015

Thanks for sharing it.

[Reply](#)[Pranas](#)

April 4, 2019

Awesome lib. Thanks Mikael!

Not sure what you learned or how chronicle inspired your. Your impl is much cleaner!

[Reply](#)

Adrien W

March 17, 2015

Thanks for the clear tutorial, it helped me a lot.

I was searching for a way to read quickly bytes in large file and I was using a ReadableByteChannel. I compared both methods and it seems that the MappedByteBuffer is faster.

[Reply](#)

Leave a Reply

Enter your comment here...

Search Tutorials

Type and Press ENTER...



Meta Links

Recommended Reading

[About Me](#)

[10 Life Lessons](#)

[Contact Us](#)

[Secure Hash Algorithms](#)

[Privacy policy](#)

[How Web Servers work?](#)

[Advertise](#)

[How Java I/O Works Internally?](#)

[Guest and Sponsored Posts](#)

[Best Way to Learn Java](#)

[Java Best Practices Guide](#)

[Microservices Tutorial](#)

[REST API Tutorial](#)

[How to Start New Blog](#)

Copyright © 2021 · Hosted on Bluehost · Sitemap