

转

Java 复制大文件方式FileChannel 用法

2015年10月13日 16:24:24

阅读数：1496

更多

目前为止，我们已经学习了很多 Java 拷贝文件的方式，除了 FileChannel 提供的方法外，还包括使用 Files.copy 方法。这个问题很难回答，因为答案基于很多因素。本文将目光集中到一个因素，那就是速度，因为拷贝任务 越快将会提高效率。下面比较下面这些拷贝方式的具体时间：

- FileChannel 和非直接模式的 ByteBuffer
- FileChannel 和直接模式的 ByteBuffer
- FileChannel.transferTo()
- FileChannel.transferFrom()
- FileChannel.map()
- 使用字节数组和缓冲流
- 使用字节数组和非缓冲流
- File.copy() (Path 到 Path , InputStream 到 Path 和 Path 到 OutputStream)

应用程序基于下面的条件：

- 拷贝文件类型 MP4 视频 (文件名为 Rafa Best Shots.mp4 , 所在目录为 C:\rafaelnadal\tournaments\2009\videos)
- 文件大小：58.3MB
- 测试的缓冲区大小：4KB, 16KB, 32KB, 64KB, 128KB, 256KB, and 1024KB
- 机器配置：Mobile AMD Sempron Processor 3400 + 1.80 GHz, 1.00GB RAM, 32-bit
- OS, Windows 7 Ultimate
- 测量类型：使用 System.nanoTime() 方法
- 连续运行三次后再获取时间；前三次运行将会被忽略。开始运行的时间总会比后面运行的时间要长一些。

下面将列出完整的应用程序：

```
1 import java.nio.MappedByteBuffer;
2 import java.io.OutputStream;
3 import java.io.InputStream;
4 import java.io.BufferedInputStream;
5
6 import java.io.File;
7 import java.io.FileInputStream;
8 import java.io.FileOutputStream;
9 import java.io.IOException;
10 import java.nio.ByteBuffer;
11 import java.nio.channels.FileChannel;
12 import java.nio.file.Files;
13 import java.nio.file.Path;
14 import java.nio.file.Paths;
15 import java.nio.file.StandardOpenOption;
16 import java.util.EnumSet;
17 import static java.nio.file.LinkOption.NOFOLLOW_LINKS;
18
19 public class Main {
20
21     public static void deleteCopied(Path path){
22
23         try {
24             Files.deleteIfExists(path);
25         } catch (IOException ex) {
26             System.err.println(ex);
27         }
28     }
29
30
31     public static void main(String[] args) {
32
33         final Path copy_from = Paths.get("C:/rafaelnadal/tournaments/2009/videos/
34                                     Rafa Best Shots.mp4");
```

0

1

使用字节数组的缓冲/非缓冲流。那个才是最好的。在某些情况下，这是成功的关键。因此，本文将使用

```

35 final Path copy_to = Paths.get("C:/Rafa Best Shots.mp4");
36 long startTime, elapsedTime;
37 int bufferSizeKB = 4; //also tested for 16, 32, 64, 128, 256 and 1024
38 int bufferSize = bufferSizeKB * 1024;
39
40 deleteCopied(copy_to);
41
42 //FileChannel and non-direct buffer
43 System.out.println("Using FileChannel and non-direct buffer ...");
44 try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
45     EnumSet.of(StandardOpenOption.READ)));
46     FileChannel fileChannel_to = (FileChannel.open(copy_to,
47     EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.WRITE))) {
48
49     startTime = System.nanoTime();
50
51     // Allocate a non-direct ByteBuffer
52     ByteBuffer bytebuffer = ByteBuffer.allocate(bufferSize);
53
54     // Read data from file into ByteBuffer
55     int bytesCount;
56     while ((bytesCount = fileChannel_from.read(bytebuffer)) > 0) {
57         //flip the buffer which set the limit to current position, and position to 0
58         bytebuffer.flip();
59         //write data from ByteBuffer to file
60         fileChannel_to.write(bytebuffer);
61         //for the next read
62         bytebuffer.clear();
63     }
64
65     elapsedTime = System.nanoTime() - startTime;
66     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
67 } catch (IOException ex) {
68     System.err.println(ex);
69 }
70
71 deleteCopied(copy_to);
72
73 //FileChannel and direct buffer
74 System.out.println("Using FileChannel and direct buffer ...");
75 try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
76     EnumSet.of(StandardOpenOption.READ)));
77     FileChannel fileChannel_to = (FileChannel.open(copy_to,
78     EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.WRITE)))) {
79
80     startTime = System.nanoTime();
81
82     // Allocate a direct ByteBuffer
83     ByteBuffer bytebuffer = ByteBuffer.allocateDirect(bufferSize);
84
85     // Read data from file into ByteBuffer
86     int bytesCount;
87     while ((bytesCount = fileChannel_from.read(bytebuffer)) > 0) {
88         //flip the buffer which set the limit to current position, and position to 0
89         bytebuffer.flip();
90         //write data from ByteBuffer to file
91         fileChannel_to.write(bytebuffer);
92         //for the next read
93         bytebuffer.clear();
94     }
95
96     elapsedTime = System.nanoTime() - startTime;

```

```

97     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
98 } catch (IOException ex) {
99     System.err.println(ex);
100 }
101
102 deleteCopied(copy_to);
103
104 //FileChannel.transferTo()
105 System.out.println("Using FileChannel.transferTo method ...");
106 try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
107     EnumSet.of(StandardOpenOption.READ)));
108     FileChannel fileChannel_to = (FileChannel.open(copy_to,
109     EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.WRITE))) {
110
111     startTime = System.nanoTime();
112
113     fileChannel_from.transferTo(0L, fileChannel_from.size(), fileChannel_to);
114
115     elapsedTime = System.nanoTime() - startTime;
116     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
117 } catch (IOException ex) {
118     System.err.println(ex);
119 }
120
121 deleteCopied(copy_to);
122
123 //FileChannel.transferFrom()
124 System.out.println("Using FileChannel.transferFrom method ...");
125 try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
126     EnumSet.of(StandardOpenOption.READ)));
127     FileChannel fileChannel_to = (FileChannel.open(copy_to,
128     EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.WRITE)))) {
129
130     startTime = System.nanoTime();
131
132     fileChannel_to.transferFrom(fileChannel_from, 0L, (int) fileChannel_from.size());
133
134     elapsedTime = System.nanoTime() - startTime;
135     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
136 } catch (IOException ex) {
137     System.err.println(ex);
138 }
139
140 deleteCopied(copy_to);
141
142 //FileChannel.map
143 System.out.println("Using FileChannel.map method ...");
144 try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
145     EnumSet.of(StandardOpenOption.READ)));
146     FileChannel fileChannel_to = (FileChannel.open(copy_to,
147     EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.WRITE)))) {
148
149     startTime = System.nanoTime();
150     MappedByteBuffer buffer = fileChannel_from.map(FileChannel.MapMode.READ_ONLY,
151         0, fileChannel_from.size());
152
153     fileChannel_to.write(buffer);
154     buffer.clear();
155
156     elapsedTime = System.nanoTime() - startTime;
157     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
158 } catch (IOException ex) {

```

```

159     System.err.println(ex);
160 }
161
162 deleteCopied(copy_to);
163
164 //Buffered Stream I/O
165 System.out.println("Using buffered streams and byte array ...");
166 File inFileStr = copy_from.toFile();
167 File outFileStr = copy_to.toFile();
168 try (BufferedInputStream in = new BufferedInputStream(new FileInputStream(inFileStr));
169      BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream(outFileStr))) {
170
171     startTime = System.nanoTime();
172
173     byte[] byteArray = new byte[bufferSize];
174     int bytesCount;
175     while ((bytesCount = in.read(byteArray)) != -1) {
176         out.write(byteArray, 0, bytesCount);
177     }
178
179     elapsedTime = System.nanoTime() - startTime;
180     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
181 } catch (IOException ex) {
182     System.err.println(ex);
183 }
184
185 deleteCopied(copy_to);
186
187 System.out.println("Using un-buffered streams and byte array ...");
188 try (FileInputStream in = new FileInputStream(inFileStr);
189      FileOutputStream out = new FileOutputStream(outFileStr)) {
190
191     startTime = System.nanoTime();
192
193     byte[] byteArray = new byte[bufferSize];
194     int bytesCount;
195     while ((bytesCount = in.read(byteArray)) != -1) {
196         out.write(byteArray, 0, bytesCount);
197     }
198
199     elapsedTime = System.nanoTime() - startTime;
200     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
201 } catch (IOException ex) {
202     System.err.println(ex);
203 }
204
205 deleteCopied(copy_to);
206
207 System.out.println("Using Files.copy (Path to Path) method ...");
208 try {
209     startTime = System.nanoTime();
210
211     Files.copy(copy_from, copy_to, NOFOLLOW_LINKS);
212
213     elapsedTime = System.nanoTime() - startTime;
214     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
215 } catch (IOException e) {
216     System.err.println(e);
217 }
218
219 deleteCopied(copy_to);
220

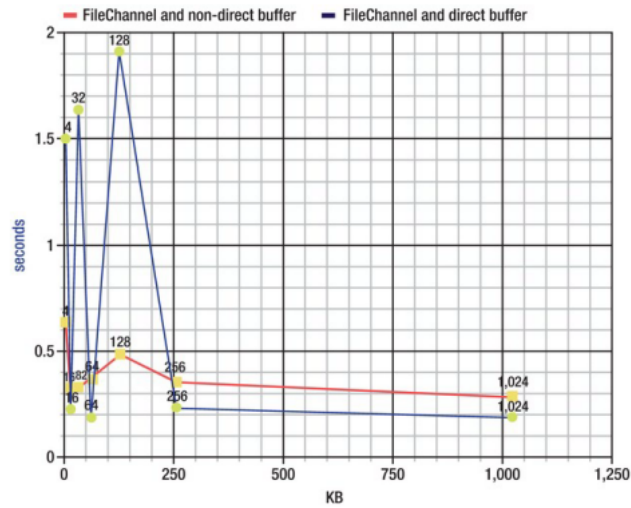
```

```
221 System.out.println("Using Files.copy (InputStream to Path) ...");
222 try (InputStream is = new FileInputStream(copy_from.toFile())) {
223
224     startTime = System.nanoTime();
225
226     Files.copy(is, copy_to);
227
228     elapsedTime = System.nanoTime() - startTime;
229     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " second
230 } catch (IOException e) {
231     System.err.println(e);
232 }
233
234 deleteCopied(copy_to);
235
236 System.out.println("Using Files.copy (Path to OutputStream) ...");
237 try (OutputStream os = new FileOutputStream(copy_to.toFile())) {
238
239     startTime = System.nanoTime();
240
241     Files.copy(copy_from, os);
242
243     elapsedTime = System.nanoTime() - startTime;
244     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
245 } catch (IOException e) {
246     System.err.println(e);
247 }
248 }
249 }
```

输出结果排序比较复杂，其中包含了很多数据。下面我将主要的对比用图形的方式展示出来。图形中 Y 坐标表示消耗的时间（单位：秒），X 坐标表示缓冲的大小（或运行前三次运行）。

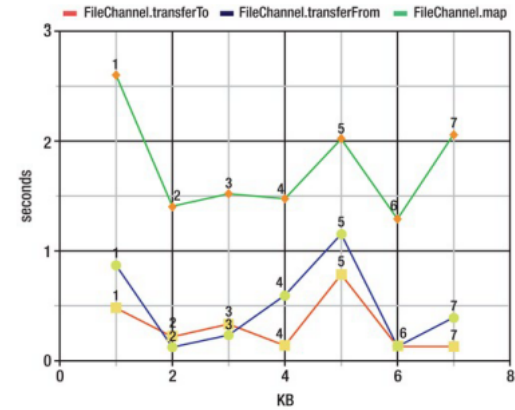
FileChannel 和非直接模式 Buffer vs. FileChannel 和直接模式 Buffer

从下图看来，如果缓存小于 256KB，那么非直接模式的 Buffer 快一点，而缓存大于 256KB 后，直接模式的 Buffer 快一点：



FileChannel.transferTo() vs. FileChannel.transferFrom() vs. FileChannel.map()

从下图看来，FileChannel.transferTo() 和 FileChannel.transferFrom 运行七次的速度都差不多，而 FileChannel.map 的速度就要差很多：



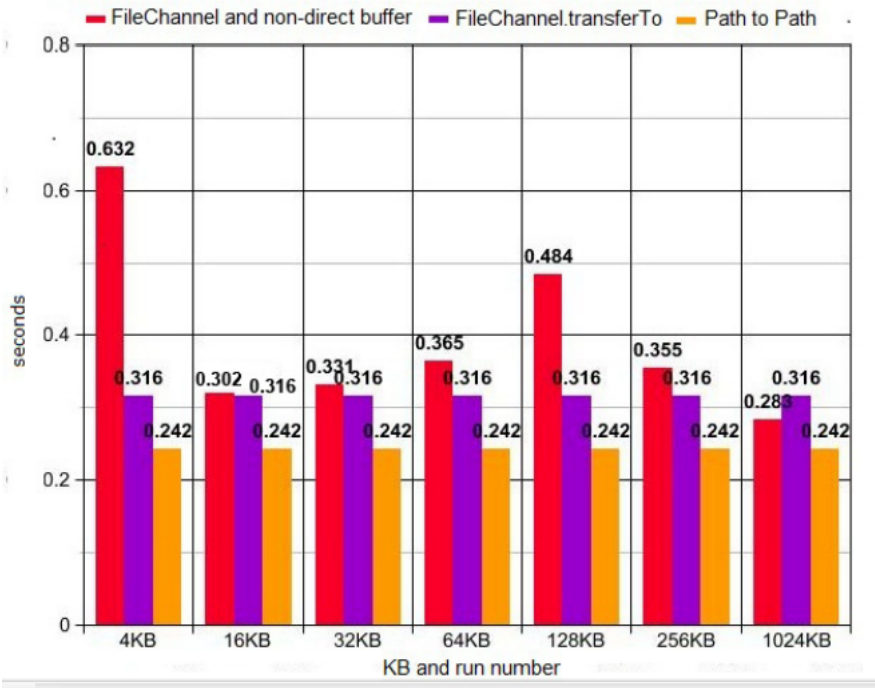
0
1

三种 Files.copy() 方法


从下图看来，最快的是 Path 到 Path，其次是 Path 到 OutputStream，最慢的是 InputStream 到 Path：


FileChannel 和非直接模式 Buffer vs. FileChannel.transferTo() vs. Path 到 Path


最后，我们将前面最快的三种方式综合起来比较。从比较的结果来看，似乎 Path 到 Path 是最快的解决方案：




 想对作者说点什么

 m0_37837862：写的不错不错 (07-21 12:13 #1楼)

不要在linux上使用java 7 Files的接口参数StandardOpenOption.DELETE_ON_CLOSE  2513
最近在看安全代码规范建议中提到关于如何删除创建的临时文件，推荐使用jdk7中的Files的函数，通过参数StandardOpenOption.DEL...

MD5文件加密以及关于NIO中的FileChannel.map的一点看法  3826
MD5文件加密以及关于NIO中的FileChannel.map的一点看法

FileChannel  696
0 概述 Channel相关知识可以参考Channel的基础。Java NIO中的FileChannel是一个连接到文件的通道。可以通过文件通道读写...

RandomAccessFile、FileChannel、MappedByteBuffer读写文件  4199
代码：package com.nio; import java.io.Closeable; import java.io.FileNotFoundException; import java.io...

[疯狂Java]NIO : Channel的map映射 1. 通道映射技术： 1) 其实就是一种快速读写技术，它将通道所连接的数据节点中的全部或部分数据直接映射到内存的一个Buffer中...	 3217
java IO相关API探索之FileChannel类 昨天看到了FileInputStream类的内部实现，里面除了继承自InputStream类外，还多加了一个FileChannel的方法，这个FileChannel()方...	 3918
Java NIO通道Channel的原理与获取 通道Channel：由java.nio.channels包定义。Channel表示IO源与目标打开的连接。Channel类似于传统的"流"，只不过Channel本身不...	 3747
java7 NIO2(4) 文件和目录操作API 文件和目录操作API，跟原来FILE IO做了很多改进，我们看看新的API，这个也是NIO操作的基础。 package com.n... nport java.io...	 1万
Java 复制大文件方式 (nio2 FileChannel 拷贝文件能力测试) 目前为止，我们已经学习了很多 Java 拷贝文件的方式，除了 FileChannel 提供的方法外，还包括使用 Files.copy() 或使用字节数组的...	 6015
相关热词 java11 与java java的~ java java和--	
JAVA之NIO按行读取大文件 做项目过程中遇到要解析100多M的TXT文件，并入库。用之前的FileInputStream、BufferedReader显然不行了，虽然readLine这方法...	 1.2万
FileChannel分割与合并文件 要求：1、将一个大文件按指定的块大小，分割成许多个小文件，每个小文件的命名，按照其大文件中的顺序从 0 开始依次递增，文件...	 128
FileChannel的深入理解 FileChannel的深入理解	 5614
FileChannel 转载自：http://ifeve.com/file-channel/ Java NIO中的FileChannel是一个连接到文件的通道。可以通过文件通道读写文件。FileChannel...	 792
利用FileChannel完成文件的读、写、复制 内容：通过NIO中的FileChannel完成文件的读、写、复制。 [java] view plain copy public class NioF...	 690
java.nio.channels.FileChannel源码解读 版本：JDK7 package java.nio.channels; import java.io.*; import java.nio.ByteBuffer; import java.nio.M...	 342
FileChannel之文件输入输出 1.文件的输入。 文件->读到文件流->文件通道->映射到内存->写入一个字符数组 File->FileInputStream->FileChannel->MappedByteBuf...	 306
Java之FileChannel类的理解和使用 Java之FileChannel类的理解和使用文章链接：知识点： FileChannel类及方法理解； 普通输入输出流复制文件； FileChannel复制文件...	 1万
使用FileChannel(文件通道)来实现文件快速复制 在Java编程中，复制文件的方法有很多，而且经常要用到。我以前一直是缓冲输入输出流来实现的（绝大多数人都是如此），近来在研...	 7128
java大文件复制最高效方法：多线程FileChannel 单线程下现在主流的复制方法有以下几种： 1、FileChannel 2、FileInputStream 3、BufferedOutputStream 4、BufferedReader 5...	 4434

个人资料



bestone0213

关注

原创

19

粉丝

127

喜欢

57

评论

56

等级：

博客 5

访问：

48万+

积分：

5520

排名：

6761

最新文章

java随机数方法解析

java邮件解析4

java邮件解析3

java邮件解析2

java邮件解析1

个人分类

opencv5篇

matlab2篇

数学理论6篇

嵌入式驱动3篇

视频编码解析2篇

展开

归档

2016年9月1篇

2016年2月5篇

2015年11月5篇

2015年10月17篇

2015年9月10篇

展开

热门文章

依赖注入和控制反转的理解，写的太好了。
阅读量：63419

entrySet用法 以及遍历map的用法
阅读量：15760

springmvc配置文件web.xml详解各方总结。
阅读量：10289

查看mysql访问记录
阅读量：8654

springmvc+Freemarker配置说明详解1
阅读量：7905

最新评论

std::vector的reser...
ADEK1NG：讲得很清楚

Java 复制大文件方式FileC...
m0_37837862：写的不错不错

依赖注入和控制反转的理解，写的太好...
qq_37821183：图pain没有

依赖注入和控制反转的理解，写的太好...
ght886：确实解释的很清楚！

依赖注入和控制反转的理解，写的太好...
yqj2065：https://blog.csdn.net/yqj2065/article/details/804...

0
1