转 Java 复制大文件方式FileChannel 用法

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.cop 使用字节数组的缓冲/非缓冲流。那个才是最好的题很难回答，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 些情况下，这是成功的关键。因此，本文将使用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：

```
import java.nio.MappedByteBuffer;
import java.io.OutputStream;
import java.io.InputStream;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.FileChannel;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.EnumSet;
import static java.nio.file.LinkOption.NOFOLLOW_LINKS;

public class Main {

 public static void deleteCopied(Path path){

  try {
      Files.deleteIfExists(path);
  } catch (IOException ex) {
    System.err.println(ex);
  }

 }

 public static void main(String[] args) {

 final Path copy_from = Paths.get("C:/rafaelnadal/tournaments/2009/videos/
                                                                Rafa Best Shots.mp4");
```

```
final Path copy_to = Paths.get("C:/Rafa Best Shots.mp4");
long startTime, elapsedTime;
int bufferSizeKB = 4; //also tested for 16, 32, 64, 128, 256 and 1024
int bufferSize = bufferSizeKB * 1024;

deleteCopied(copy_to);

//FileChannel and non-direct buffer
System.out.println("Using FileChannel and non-direct buffer ...");
try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
                    EnumSet.of(StandardOpenOption.READ)));
     FileChannel fileChannel_to = (FileChannel.open(copy_to,
                    EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.WRITE))) {

     startTime = System.nanoTime();

     // Allocate a non-direct ByteBuffer
     ByteBuffer bytebuffer = ByteBuffer.allocate(bufferSize);

     // Read data from file into ByteBuffer
     int bytesCount;
     while ((bytesCount = fileChannel_from.read(bytebuffer)) > 0) {
      //flip the buffer which set the limit to current position, and position to 0
      bytebuffer.flip();
      //write data from ByteBuffer to file
      fileChannel_to.write(bytebuffer);
      //for the next read
      bytebuffer.clear();
     }

     elapsedTime = System.nanoTime() - startTime;
     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
} catch (IOException ex) {
  System.err.println(ex);
}

deleteCopied(copy_to);

//FileChannel and direct buffer
System.out.println("Using FileChannel and direct buffer ...");
try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
                    EnumSet.of(StandardOpenOption.READ)));
     FileChannel fileChannel_to = (FileChannel.open(copy_to,
                    EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.WRITE)))) {

     startTime = System.nanoTime();

     // Allocate a direct ByteBuffer
     ByteBuffer bytebuffer = ByteBuffer.allocateDirect(bufferSize);

     // Read data from file into ByteBuffer
     int bytesCount;
     while ((bytesCount = fileChannel_from.read(bytebuffer)) > 0) {
      //flip the buffer which set the limit to current position, and position to 0
      bytebuffer.flip();
      //write data from ByteBuffer to file
      fileChannel_to.write(bytebuffer);
      //for the next read
      bytebuffer.clear();
     }

     elapsedTime = System.nanoTime() - startTime;
```

```
        System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
    } catch (IOException ex) {
      System.err.println(ex);
    }

    deleteCopied(copy_to);

    //FileChannel.transferTo()
    System.out.println("Using FileChannel.transferTo method ...");
    try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
                        EnumSet.of(StandardOpenOption.READ)));
        FileChannel fileChannel_to = (FileChannel.open(copy_to,
                        EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.W))) {

        startTime = System.nanoTime();

        fileChannel_from.transferTo(0L, fileChannel_from.size(), fileChannel_to);

        elapsedTime = System.nanoTime() - startTime;
        System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
    } catch (IOException ex) {
      System.err.println(ex);
    }

    deleteCopied(copy_to);

    //FileChannel.transferFrom()
    System.out.println("Using FileChannel.transferFrom method ...");
    try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
                        EnumSet.of(StandardOpenOption.READ)));
        FileChannel fileChannel_to = (FileChannel.open(copy_to,
                        EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.WRITE)))) {

        startTime = System.nanoTime();

        fileChannel_to.transferFrom(fileChannel_from, 0L, (int) fileChannel_from.size());

        elapsedTime = System.nanoTime() - startTime;
        System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
    } catch (IOException ex) {
      System.err.println(ex);
    }

    deleteCopied(copy_to);

    //FileChannel.map
    System.out.println("Using FileChannel.map method ...");
    try (FileChannel fileChannel_from = (FileChannel.open(copy_from,
                        EnumSet.of(StandardOpenOption.READ)));
        FileChannel fileChannel_to = (FileChannel.open(copy_to,
                        EnumSet.of(StandardOpenOption.CREATE_NEW, StandardOpenOption.WRITE)))) {

        startTime = System.nanoTime();
        MappedByteBuffer buffer = fileChannel_from.map(FileChannel.MapMode.READ_ONLY,
                                                       0, fileChannel_from.size());

        fileChannel_to.write(buffer);
        buffer.clear();

        elapsedTime = System.nanoTime() - startTime;
        System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
    } catch (IOException ex) {
```

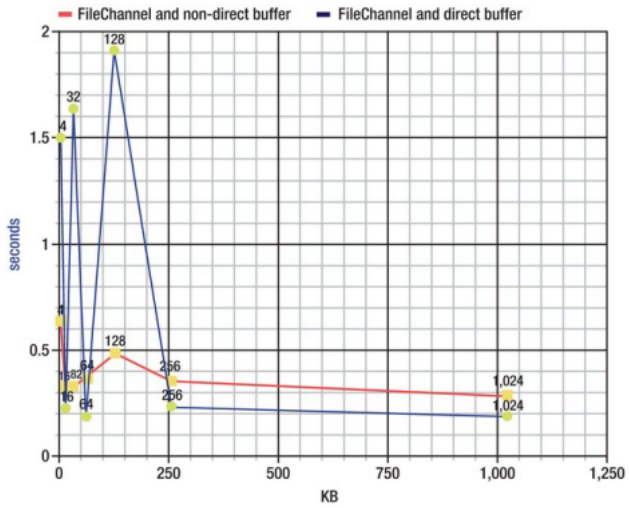
```
  System.err.println(ex);
}

deleteCopied(copy_to);

//Buffered Stream I/O
System.out.println("Using buffered streams and byte array ...");
File inFileStr = copy_from.toFile();
File outFileStr = copy_to.toFile();
try (BufferedInputStream in = new BufferedInputStream(new FileInputStream(inFileStr
     BufferedOutputStream out = new BufferedOutputStream(new FileOutputStream(outFi      )) {

     startTime = System.nanoTime();

     byte[] byteArray = new byte[bufferSize];
     int bytesCount;
     while ((bytesCount = in.read(byteArray)) != -1) {
             out.write(byteArray, 0, bytesCount);
     }

     elapsedTime = System.nanoTime() - startTime;
     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
} catch (IOException ex) {
  System.err.println(ex);
}

deleteCopied(copy_to);

System.out.println("Using un-buffered streams and byte array ...");
try (FileInputStream in = new FileInputStream(inFileStr);
     FileOutputStream out = new FileOutputStream(outFileStr)) {

     startTime = System.nanoTime();

     byte[] byteArray = new byte[bufferSize];
     int bytesCount;
     while ((bytesCount = in.read(byteArray)) != -1) {
             out.write(byteArray, 0, bytesCount);
     }

     elapsedTime = System.nanoTime() - startTime;
     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
} catch (IOException ex) {
  System.err.println(ex);
}

deleteCopied(copy_to);

System.out.println("Using Files.copy (Path to Path) method ...");
try {
    startTime = System.nanoTime();

    Files.copy(copy_from, copy_to, NOFOLLOW_LINKS);

    elapsedTime = System.nanoTime() - startTime;
    System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
} catch (IOException e) {
  System.err.println(e);
}

deleteCopied(copy_to);
```

```
System.out.println("Using Files.copy (InputStream to Path) ...");
try (InputStream is = new FileInputStream(copy_from.toFile())) {

    startTime = System.nanoTime();

    Files.copy(is, copy_to);

    elapsedTime = System.nanoTime() - startTime;
    System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " second
} catch (IOException e) {
  System.err.println(e);
}

deleteCopied(copy_to);

System.out.println("Using Files.copy (Path to OutputStream) ...");
try (OutputStream os = new FileOutputStream(copy_to.toFile())) {

     startTime = System.nanoTime();

     Files.copy(copy_from, os);

     elapsedTime = System.nanoTime() - startTime;
     System.out.println("Elapsed Time is " + (elapsedTime / 1000000000.0) + " seconds");
 } catch (IOException e) {
   System.err.println(e);
 }
 }
}
```

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）。

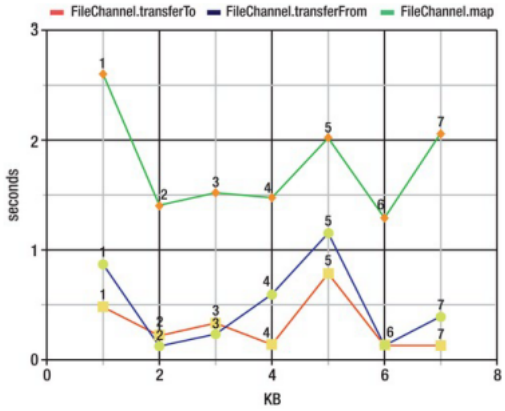
FileChannel 和非直接模式 Buffer vs. FileChannel 和直接模式 Buffer

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：

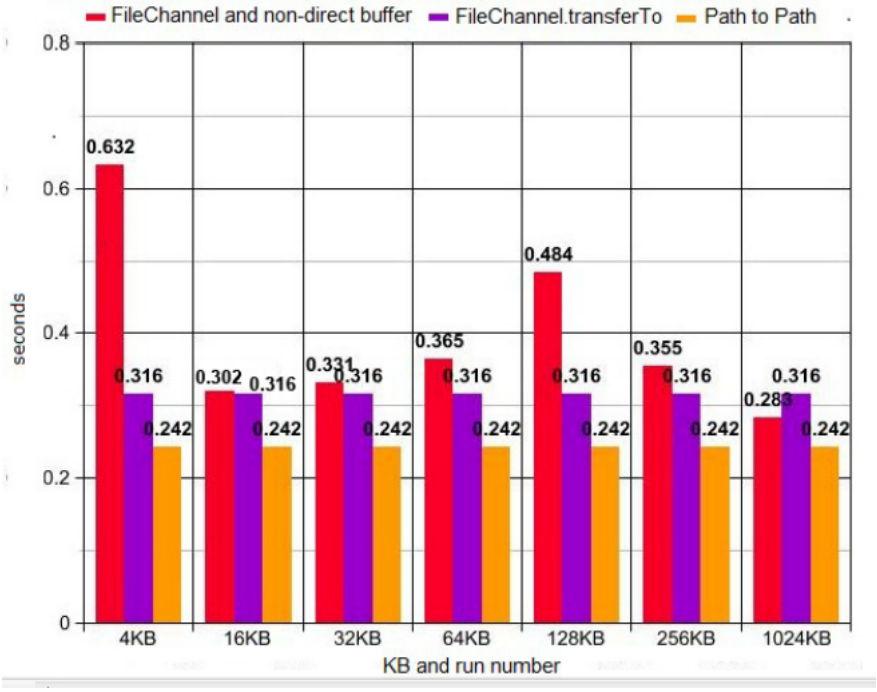


三种 Files.copy() 方法

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：



[疯狂Java]NIO：Channel的map映射 3217

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 个Buffer中…

0

java IO相关API探索之FileChannel类 3918

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，通 Channel()方…

1

Java NIO通道Channel的原理与获取 3747

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 nnel本身不…

java7 NIO2(4) 文件和目录操作API 1万

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…

Java 复制大文件方式（nio2 FileChannel 拷贝文件能力测试） 6015

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…

相关热词 javall 与java java的~ java java和--

JAVA之NIO按行读取大文件 1.2万

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…

FileChannel分割与合并文件 128

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…

FileChannel的深入理解 5614

FileChannel的深入理解

FileChannel 792

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…

利用FileChannel完成文件的读、写、复制 690

内容：通过NIO中的FileChannel完成文件的读、写、复制。 [java] view plain copy public class NioF...

java.nio.channels.FileChannel源码解读 342

版本：JDK7 package java.nio.channels; import java.io.*; import java.nio.ByteBuffer; import java.nio.M...

FileChannel之文件输入输出 306

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…

Java之FileChannel类的理解和使用 1万

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…

使用FileChannel(文件通道)来实现文件快速复制 7128

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…

java大文件复制最高效方法：多线程FileChannel 4434

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...

个人资料

bestone0213 关注

原创	粉丝	喜欢	评论
19	127	57	56