在之前的章节我们已经顺序地探索了文件.文件可以被顺序探索也被称为有序的文件.在这章,你将看到使用无序(随机)访问一个文件内容的优势.文件允许随机范文它们的内容也被称为随机访问文件(RAFs).有序的文件更被经常使用因为它们容易去创建,但是RAFs更加灵活,它们的数据可被快速地定位.

使用RAF,你可以打开文件,搜索一个特殊的位置,从它读取或者写入.在你打开一个RAF后,你可以从它读取或者通过使用一个记录编号以一种随机的方式写入,或者你可以加入到文件的开始或者结尾,因为你知道在文件中有多少记录.一个RAF允许你读取一个单独的字符,读取一个块的字节或者一行,替换文件的一部分,追加行,删除行,等等,允许你以一种随机方式执行所有的这些动作.

Java 7(NIO.2)引入了一个崭新的(brand-new)接口处理RAFs.它的名称为SeekableByteChannel,它在java.nio.channels包中可用.它集成了老的ByteChannel接口,代表了一个字节通道操纵一个当前位置和允许位置被修改.此外,Java 7一次性改善了总所周知的FileChannel类,通过实现这个接口.提供了RAF和FileChannel的能力.通过一个简单的转型,我们可以转换一个SeekableByteChannel为FileChannel.

这章广泛地(extensively)使用了java.nio.ByteBuffer类,所以我们以它的一个简单的概览开始.我们将继续通过应用程序细节化SeekableByteChannel接口,它将随机地读取和写入文件区完成不同类型的共同的任务.你将看到如何去使用RAF的能力获取FileChannel,浏览通过FileChannel提供的主要的功能,比如直接映射文件的一个部分到内存为了快速访问,锁定文件的一个部分,从一个绝对位置读取和写入字节而不会影响通道的当前的位置.这章以一个标记(FIXME:benchmarking)应用程序结束,这将帮助你使用FileChannel的能力去决定最快速的方式去拷贝一个文件,与其它的方法相比,像Files.copy(),缓冲的流,等等.

ByteBuffer的简要概览

一个缓冲区本质上是一个数组(通常为字节,但是其它类型的数组可以被使用-Buffer接口提供了ByteBuffer,CharBuffer,IntBuffer,ShortBuffer,LongBuffer,FloatBuffer和DoubleBuffer)保存一些数据只有在读的时候才写入.

在NIO中两个最重要的缓冲区组件是属性和父类的方法(FIXME:ancestor method),依次在下面讨论.

ByteBuffer属性

下面是一个缓冲区的基本属性:

- Limit(极限):当从一个缓冲区读取的时候,极限指定了还剩余多少数据要获取.当你向一个缓冲区写入的时候,极限指定了还有多少剩余数据需要被放入(这里书中前后的意思写反了,应该是When reading from a buffer, the limit specifies how much data remains to get. When you are writing into a buffer, the limit specifies how much room remains to put data into).
- Position(位置):位置追踪你已经读取或者写入了多少数据.它指定了数组元素的下一个字节将出去或者进来.一个缓冲区的位置永远不会是负数,并且永远不大于它的极限.
- Capacity(容量):容量指定了存储在一个缓冲区最大的数据的数目.极限永远不能大于容量.

注意:作为一个不变式,这些属性遵循下面的关系:0 ≤ position ≤ limit ≤ capacity.

作为一个示例,假设一个缓冲区有6个字节容量,如图7-1所示.

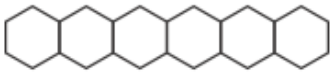

Figure 7-1. *Java buffer representation (a)*

在一开始,极限和容量是相等的(极限不能大于容量,但是相反是完成正常的),设置在一个虚拟的位置(在我们的例子中,槽位号为7),如图7-2所示.

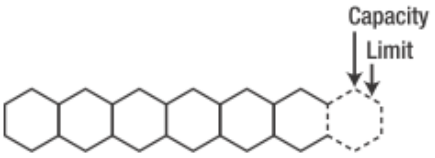

Figure 7-2. *Java buffer representation (b)*

注意:在一些情况下,初始的极限可能为0,或者它肯能是其它值,依赖于缓冲区的类型和它被构造的方式.

同样,在一开始,位置设置为0(槽位1,如图7-3)-一个读取字节或者写入字节将访问位置0.

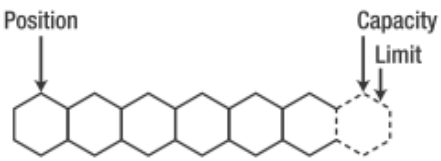

Figure 7-3. *Java buffer representation (c)*

往下,假设我们往缓冲区写入两个字节的数据.2个字节的数据将从缓冲区的位置0开始进入.因此,2个字节将被填充,位置将到第三个字节,如图7-4所示.
(注意:这里的表述有点混乱,混淆了read和write,这里应该是write,后面才是read)

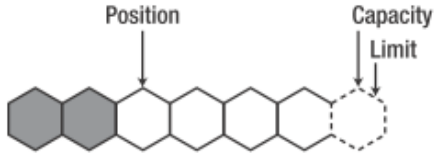

Figure 7-4. *Java buffer representation (d)*

继续第二次写入,另外三个字节进入缓冲区.位置增长到5(槽位6),如你在图7-5中看到的.

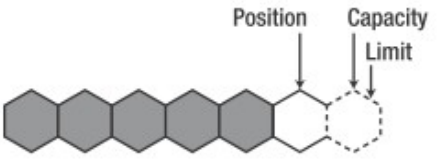

Figure 7-5. *Java buffer representation (e)*

现在,假设我们不再往缓冲区中写入数据,而是想从缓冲区中读取数据.为了这样,我们首先需要

去调用flip()方法在我们读取任意字节前.这个将设置极限为当前的位置,设置位置为0.在flip之后,缓冲区如图9-6所示.

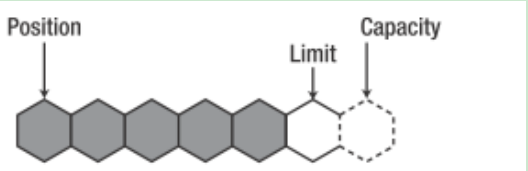

Figure 7-6. *Java buffer representation (f)*

加上我们从缓冲区中读取3个字节.因为位置为0,头3个字节被读取,位置移动到3(槽位4),如图7-7所示.极限和容量保持不变.

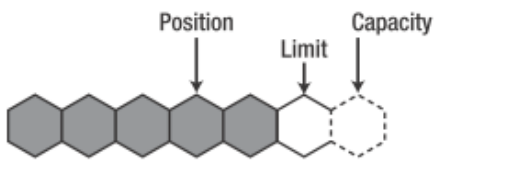

Figure 7-7. *Java buffer representation (g)*

下面,我们读取2个字节,位置移动转到槽位6,如图7-8所示;极限和容量保持不变.

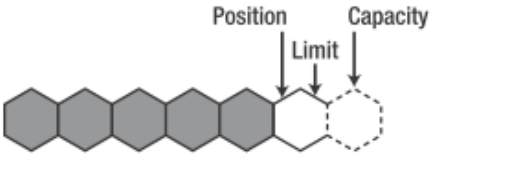

Figure 7-8. *Java buffer representation (h)*

有两个额外的操作我们可能想要完成.继续使用图7-8作为引用,我们想去反转(rewind)缓冲区或者清除缓冲区.反转缓冲区(调用rewind()方法)为缓冲区准备好重新读取它已经包含的数据-极限保持不变,位置设置为0.清除缓冲区(调用clear()方法)将重置缓冲区来接收更多的字节(数据没有被删除)-极限设置为容量,位置设置为0.图7-9展示了clear()方法的效果,图7-10展示了rewind()方法的效果.

Figure 7-9. *Java buffer representation (i)*

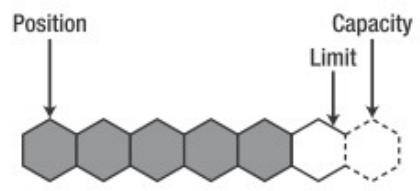

Figure 7-10. *Java buffer representation (j)*

另外,一个缓冲器持有一个mark(标记).这是将要被重置的位置的索引当reset()方法被调用的时候.mark总是未定义的,但是它从不为负数,从不大于位置.如果标记被定义,它将被废弃当位置或者极限调整为一个比标记小的值.如果标记未定义,调用reset()方法将抛出InvalidMarkException.

注意:插入标记到下面的关系结果中: 0 ≤ mark ≤ position ≤ limit ≤ capacity.

ByteBuffer父类方法

ByteBuffer提供了一组get()和put()方法用来访问数据.因此它们是相当直观的(intuitive).我将简单的将它们列在这里.获取更多的细节,查阅官方文档:http://download.oracle.com/javase/7/docs/api/index.html和http://download.oracle.com/javase/7/docs/index.html.

```
public abstract byte get()
public ByteBuffer get(byte[] dst)
public ByteBuffer get(byte[] dst, int offset, int length)
public abstract byte get(int index)

public abstract ByteBuffer put(byte b)
public final ByteBuffer put(byte[] src)
public ByteBuffer put(byte[] src, int offset, int length)
public ByteBuffer put(ByteBuffer src)
public abstract ByteBuffer put(int index, byte b)
```

除了get()和put()方法,ByteBuffer也有一些额外的方法用来读取和写入不同类型的值,如下:

```
public abstract char getChar()
public abstract char getChar(int index)
public abstract double getDouble()
public abstract double getDouble(int index)
public abstract float getFloat()
public abstract float getFloat(int index)
public abstract int getInt()
public abstract int getInt(int index)
public abstract long getLong()
public abstract long getLong(int index)
public abstract short getShort()
public abstract short getShort(int index)

public abstract ByteBuffer putChar(char value)
public abstract ByteBuffer putChar(int index, char value)
public abstract ByteBuffer putDouble(double value)
public abstract ByteBuffer putDouble(int index, double value)
public abstract ByteBuffer putFloat(float value)
public abstract ByteBuffer putFloat(int index, float value)
public abstract ByteBuffer putInt(int value)
public abstract ByteBuffer putInt(int index, int value)
```

```
public abstract ByteBuffer putLong(int index, long value)
public abstract ByteBuffer putLong(long value)
public abstract ByteBuffer putShort(int index, short value)
public abstract ByteBuffer putShort(short value)
```

一个字节缓冲区可以是直接或者非直接的.JVM将在直接缓冲区上执行本地I/O操作.直接缓冲区通过使用allocateDirect()方法创建,非直接缓冲区通过使用allocate()方法创建.

此时,你已经有了关于ByteBuffer足够的信息区理解下面的应用程序(为了深入ByteBuffer的内部,在Web上访问专门的教程).因此,我们遗留(leave behind)ByteBuffer一会儿,处理这章的主要的主题,SeekableByteChannel接口.下个章节我们将向你介绍通道和与它们关联的缓冲区.

通道的简要概览

在一个面向流的I/O系统中,一个输入流产生1个字节的数据和一个输出流消费一个字节的数据-这样的系统通常很慢的.相比之下,在一个面向块的I/O系统中,输入/输出流在一个步骤中产生或者消费一个数据块.

通道类似于(analogous)流,但有一些区别:

- 流通常是单向的(读或写),通道支持读和写.
- 通道可以异步地读和写.
- 通道总是读取或写入一个缓冲区.所有发送到一个通道的数据必须首先存储在一个缓冲区中.从一个通道读取的任意数据写入到一个缓冲区中.(这里也是read,write混淆了)

使用SeekableByteChannel接口去随机访问文件

新的SeekableByteChannel接口为RAF提供了支持通过在通道上实现位置的概念.我们可以从一个通道中读取一个ByteBuffer或者向通道中写入一个ByteBuffer,获取或者设置当前的位置,截断连接到一个通道上的实体为一个指定的尺寸(FIXME: truncate an entity connected to a channel to a specified dimension).下面的方法与这些特性相关联(更多细节在官方文档http://download.oracle.com/javase/7/docs/api/index.html):

- position(): 返回通道当前的位置(非负数).
- position(long): 设置通道的位置为指定的long(非负数).设置位置为一个比当前大小大的值是合法的,但是不会改变实体的大小.
- truncate(long): 截取到通道的实体连接为指定的long.
- read(ByteBuffer): 从通道中读取字节到缓冲区中(对于通道,是进行读取操作,而对于缓冲区,则是往里面填充数据).
- write(ByteBuffer): 往通道中写入缓冲区中的数据.
- size(): 返回连接的通道的实体大小.

获取一个SeekableByteChannel的实例可以通过Files类,名为newByteChannel()的两个方法完成.第一个(简单的)newByteChannel方法接收到一个文件的路径去打开或者创建,有一组选项指定文件如何被打开.StandardOpenOption枚举常量在第4章"使用标准的打开选项"小结描述过,但是它们在这里重述一下用来简单的参考:

READ	为了读取访问打开文件

WRITE	为了写入访问打开文件
CREATE	创建一个新的文件如果它不存在
CREATE_NEW	创建一个新的文件,如果文件已经存在以一个异常失败
APPEND	追加文件到文件的末尾(和WEITE和CREATE一起使用)
DELETE_ON_CLOSE	删除文件当流关闭的时候(用来删除临时文件)
TRUNCATE_EXISTING	截断文件为0个字节(和WEITE一起使用)
SPARSE	使新创建的文件是sparse(FIXME)
SYNC	使用底层的存储设备保持文件内容和元数据同步
DSYNC	使用底层的存储设备保持文件内容同步

第二个newByteChanel()方法接收要打开或者创建的文件的路径,一组选项指定文件如何被打开,是可选的,一组文件属性将自动被设置当文件被创建的时候.

这两个方法打开或者创建文件,返回一个SeekableByteChannel去访问文件.

使用SeekableByteChannel读取一个文件

关注于第一个newByteChannel()方法,我们获取一个SeekableByteChannel用来读取路径C:\rafaelnadal\grandslam\RolandGarros\story.txt(文件必须存在):

```
…
Path path = Paths.get("C:/rafaelnadal/grandslam/RolandGarros", "story.txt");
…
try (SeekableByteChannel seekableByteChannel = Files.newByteChannel(path,
EnumSet.of(StandardOpenOption.READ))) {
…
} catch (IOException ex) {
   System.err.println(ex);
}
```

作为一个示例,下面的应用程序将使用ByteBuffer读取和展示story.txt的内容(文件必须存在).我选择了一个12个字节的缓冲区,但是可以随意(feel free)使用任意其它的大小.

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.SeekableByteChannel;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.EnumSet;
import java.nio.file.StandardOpenOption;
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/grandslam/RolandGarros", "story.txt");
        //read a file using SeekableByteChannel
        try (SeekableByteChannel seekableByteChannel = Files.newByteChannel(path,
                                                       EnumSet.of(StandardOpenOption.READ)))
{
            ByteBuffer buffer = ByteBuffer.allocate(12);
            String encoding = System.getProperty("file.encoding");
            buffer.clear();
            while (seekableByteChannel.read(buffer) > 0) {
```

```
                buffer.flip();
                System.out.print(Charset.forName(encoding).decode(buffer));
                buffer.clear();
            }
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

输出应该类似于下面：

```
Rafa Nadal produced another masterclass of clay-court tennis to win his
fifth French Open title ...
```

使用SeekableByteChannel写一个文件

使用SeekableByteChannel写一个文件包含了使用WRITE选项.另外,如果我们想在写入之前处理已存在的内容,我们可以加上TRUNCATE_EXISTING选项,如下.这里我们截断story.txt然后准备用它来进行写(story.txt文件必须存在).

```
…
Path path = Paths.get("C:/rafaelnadal/grandslam/RolandGarros", "story.txt");
…
try (SeekableByteChannel seekableByteChannel = Files.newByteChannel(path,
                EnumSet.of(StandardOpenOption.WRITE, StandardOpenOption.TRUNCATE_EXISTING)))
{
…
} catch (IOException ex) {
   System.err.println(ex);
}
```

作为一个示例,下面的应用程序将使用ByteBuffer在story.txt中截断和写入文本(在这种情况文件已经存在;如果它不存在,我们应该加入CREATE或者CREATE_NEW和WRITE选项和取出TRUNCATE_EXISTING选项,因为文件无论如何都是空的):

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.SeekableByteChannel;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.EnumSet;
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/grandslam/RolandGarros", "story.txt");
        //write a file using SeekableByteChannel
        try (SeekableByteChannel seekableByteChannel = Files.newByteChannel(path,
                    EnumSet.of(StandardOpenOption.WRITE,
StandardOpenOption.TRUNCATE_EXISTING))) {
            ByteBuffer buffer = ByteBuffer.wrap("Rafa Nadal produced another masterclass of
```

```
clay-courttennis to win his fifth French Open title ...".getBytes());
            int write = seekableByteChannel.write(buffer);
            System.out.println("Number of written bytes: " + write);
            buffer.clear();
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

当你写一个文件,一些常见的情况下包含了组合open选项:

- 为了写入一个存在的文件,在开始的时候,使用WRITE
- 为了写入一个存在的文件,在结束的时候,使用WRITE和APPEND.
- 为了写入一个存在的文件和在写之前清除它的内容,使用WRITE和TRUNCATE_EXISTING.
- 为了写入一个不存在的文件,使用CREATE(或者CREATE_NEW)和WRITE.

SeekableByteChannel和文件属性

下面的代码片段(为Unix和其它的POSIX文件体统写的)使用指定的一组文件权限创建文件.这个代码在home\rafaelnadal\email目录下创建了email.txt文件或者添加到它如果已经存在的话.email.txt文件被创建对于所有者和只读权限组拥有read和write权限.

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.SeekableByteChannel;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.nio.file.attribute.FileAttribute;
import java.nio.file.attribute.PosixFilePermission;
import java.nio.file.attribute.PosixFilePermissions;
import java.util.EnumSet;
import java.util.Set;
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("home/rafaelnadal/email", "email.txt");
        ByteBuffer buffer = ByteBuffer.wrap("Hi Rafa, I want to congratulate you for the
amazing match that you played ... ".getBytes());
        //create the custom permissions attribute for the email.txt file
        Set<PosixFilePermission> perms = PosixFilePermissions.fromString("rw-r------");
        FileAttribute<Set<PosixFilePermission>> attr =
PosixFilePermissions.asFileAttribute(perms);
        //write a file using SeekableByteChannel
        try (SeekableByteChannel seekableByteChannel = Files.newByteChannel(path,
                        EnumSet.of(StandardOpenOption.CREATE, StandardOpenOption.APPEND),
attr)) {
            int write = seekableByteChannel.write(buffer);
            System.out.println("Number of written bytes: " + write);
        } catch (IOException ex) {
            System.err.println(ex);
        }
        buffer.clear();
    }
```

```
}
```

使用老的ReadableByteChannel接口读取一个文件

SeekableByteChannel接口基于老的接口ReadableByteChannel(代表了一个通道读取字节;在同一时刻只能有1个线程读取)和WriteableByteChannel(代表了一个通道写入字节;在同一时刻只能有1个线程写入),它们从JDK 1.4已经在NIO中可用.这两个接口是SeekableByteChannel的父接口.由于这个关系,在它们之间,我们可以和Files.newByteChannel()方法一起使用ReadableByteChannel接口,如下,我们从存在的story.txt文件中读取内容:

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.ReadableByteChannel;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/grandslam/RolandGarros", "story.txt");
        //read a file using ReadableByteChannel
        try (ReadableByteChannel readableByteChannel = Files.newByteChannel(path)) {
            ByteBuffer buffer = ByteBuffer.allocate(12);
            buffer.clear();
            String encoding = System.getProperty("file.encoding");
            while (readableByteChannel.read(buffer) > 0) {
                  buffer.flip();
                  System.out.print(Charset.forName(encoding).decode(buffer));
                  buffer.clear();
            }
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

如你看到的,没有必要指定READ选项.

使用老的WritableByteChannel接口写一个文件

我们也可以使用新的Files.newByteChannel()方法组合老的WritableByteChannel接口,如下,我们往story.txt中追加文本:

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.channels.WritableByteChannel;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;
import java.util.EnumSet;
public class Main {
    public static void main(String[] args) {
```

```
        Path path = Paths.get("C:/rafaelnadal/grandslam/RolandGarros", "story.txt");
        //write a file using WritableByteChannel
        try (WritableByteChannel writableByteChannel =
Files.newByteChannel(path, EnumSet.of(StandardOpenOption.WRITE, StandardOpenOption.APPEND)))
{
            ByteBuffer buffer = ByteBuffer.wrap("Vamos Rafa!".getBytes());
            int write = writableByteChannel.write(buffer);
            System.out.println("Number of written bytes: " + write);
            buffer.clear();
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

即使我们使用了WritableByteChannel,我们仍然需要去明确指定WEITE选项.APPEND选项是可选的,在之前的示例中指定过.

玩转SeekableByteChannel位置(FIXME: Playing with SeekableByteChannel Position)

现在,你知道了如何使用SeekableByteChannel去读取和写入整个实体文件,你准备去覆盖,你如何做同样的操作,但是在一个指定的通道(实体)位置.为了这个,我们将使用position()和position(long)方法在一套四个示例中打算去让你熟悉RAF的概念.记住没有参数的position()方法返回当前通道(实体)的位置,position(long)方法在通道(实体)中设置当前的位置通过从它的开始增长字节的数目.第一个位置为0,最后一个有效的位置是通道(实体)的大小(FIMXE: 应该是大小-1).

示例1: 从不同的位置读取一个字符

我们从一个简单的例子开始,从一个文件的开始、中间和最后的位置准备的读取一个字符.这个文件是MovistarOpen.txt,它位于 C:\rafaelnadal\tournaments\2009 目录.

```
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/tournaments/2009", "MovistarOpen.txt");
        ByteBuffer buffer = ByteBuffer.allocate(1);
        String encoding = System.getProperty("file.encoding");
        try (SeekableByteChannel seekableByteChannel = (Files.newByteChannel(path,
EnumSet.of(StandardOpenOption.READ)))) {
            //the initial position should be 0 anyway
            seekableByteChannel.position(0);
            System.out.println("Reading one character from position: " +
 seekableByteChannel.position());
            seekableByteChannel.read(buffer);
            buffer.flip();
            System.out.print(Charset.forName(encoding).decode(buffer));
            buffer.rewind();
            //get into the middle
            seekableByteChannel.position(seekableByteChannel.size() / 2);
            System.out.println("\nReading one character from position: " +
seekableByteChannel.position());
            seekableByteChannel.read(buffer);
            buffer.flip();
            System.out.print(Charset.forName(encoding).decode(buffer));
```

```
            buffer.rewind();
            //get to the end
            seekableByteChannel.position(seekableByteChannel.size()-1);
            System.out.println("\nReading one character from position: " +
seekableByteChannel.position());

            seekableByteChannel.read(buffer);
            buffer.flip();
            System.out.print(Charset.forName(encoding).decode(buffer));
            buffer.clear();
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

这个应用程序将产生下面的输出：

```
Reading one character from position: 0
T
Reading one character from position: 181
n
Reading one character from position: 361
.
```

示例2：在不同的位置写入字符

下面,我们将尝试在不同的位置写入.假设MovistarOpen.txt文件有如下的默认内容：

```
The Movistar Open moved to Santiago from Viña del Mar in 2010. It is the
first clay-court
tournament of the ATP World Tour season and also the opening leg of the
four-tournament swing
through Latin America, aptly coined the "Golden Swing" in honour of top
Chileans and Olympic
Gold medalists Fernando Gonsales and Nicolas Massu. Gonzalez is a four-
time champion.
```

我们想去完成两个任务：第一,在之前的文本的末尾加入一些文本,第二,用"Gonzalez"替换"Gonsales", 因为Fernando的最后的名字在第一个实例中拼错了.这里是应用程序：

```
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/tournaments/2009", "MovistarOpen.txt");
        ByteBuffer buffer_1 = ByteBuffer.wrap("Great players participate in our tournament,
like: Tommy Robredo, Fernando Gonzalez, Jose Acasuso or Thomaz Bellucci.".getBytes());
        ByteBuffer buffer_2 = ByteBuffer.wrap("Gonzalez".getBytes());

        try (SeekableByteChannel seekableByteChannel = (Files.newByteChannel(path,
EnumSet.of(StandardOpenOption.WRITE)))) {
```

```
            seekableByteChannel.position(seekableByteChannel.size());

            while (buffer_1.hasRemaining()) {
                seekableByteChannel.write(buffer_1);
            }

            seekableByteChannel.position(301);

            while (buffer_2.hasRemaining()) {
                seekableByteChannel.write(buffer_2);
            }

            buffer_1.clear();
            buffer_2.clear();

        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

如果一切工作没问题,新的MovistarOpen.txt的内容应该如下:

```
The Movistar Open moved to Santiago from Viña del Mar in 2010. It is the
first clay-court tournament of the ATP World Tour season and also the
opening leg of the four-tournament swing through Latin America, aptly
coined the "Golden Swing" in honour of top Chileans and Olympic Gold
medalists Fernando Gonzalez and Nicolas Massu. Gonzalez is a four-time
champion. Great players participate in our tournament, like: Tommy
Robredo, Fernando Gonzalez, Jose Acasuso or
Thomaz Bellucci.
```

示例3：拷贝文件的一部分从开始到结束

转到一个新的应用程序,我们接下来想从文件的开始到相同文件的结束的文本的一部分的拷贝.作为一个示例,我们将使用HeinekenOpen.txt文件(位于C:\rafaelnadal\tournaments\2009目录),有如下的内容:

```
The Pride Of New Zealand
The Heineken Open is the biggest men's professional sporting event in
New Zealand, held in...
```

我们想拷贝"The Pride of New Zealand"文本到末尾,像这样:

```
The Pride Of New Zealand
The Heineken Open is the biggest men's professional sporting event in
New Zealand, held in...
The Pride Of New Zealand
```

下面的应用程序完成这个任务:

```
public class Main {
```

```
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/tournaments/2009", "HeinekenOpen.txt");

        ByteBuffer copy = ByteBuffer.allocate(25);
        copy.put("\n".getBytes());
        try (SeekableByteChannel seekableByteChannel = (Files.newByteChannel(path,
EnumSet.of(StandardOpenOption.READ, StandardOpenOption.WRITE)))) {
            int nbytes;
            do {
              nbytes = seekableByteChannel.read(copy);
            } while (nbytes != -1 && copy.hasRemaining());

            copy.flip();
            seekableByteChannel.position(seekableByteChannel.size());
            while (copy.hasRemaining()) {
                  seekableByteChannel.write(copy);
            }
            copy.clear();
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

示例4：使用截断能力替换一个文件的一部分

在这章我们将截断一个文件,然后追加新的文本替换截断的文本.我们将使用BrasilOpen.txt文件(在C:\rafaelnadal\tournaments\2009 目录中找到),这个文件有如下的内容:

```
Brasil Open At Forefront Of Green Movement
The Brasil Open, the second stop of the four-tournament Latin American
swing, is held in an
area renowned for its lush natural beauty and stunning beaches. From
this point forward ...
```

我们想去截断文件的内容,移除"From this point forward ..."然后在它的位置追加新的文本.这里是解决方案:

```
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/tournaments/2009", "BrasilOpen.txt");
        ByteBuffer buffer = ByteBuffer.wrap("The tournament has taken a lead in
environmental conservation efforts, with highlights including the planting of 500 trees to
neutralise carbon emissions and providing recyclable materials to local children for use in
craft work.".getBytes());

        try (SeekableByteChannel seekableByteChannel = (Files.newByteChannel(path,
EnumSet.of(StandardOpenOption.READ, StandardOpenOption.WRITE)))) {
            seekableByteChannel.truncate(200);
            seekableByteChannel.position(seekableByteChannel.size()-1);
            while (buffer.hasRemaining()) {
                seekableByteChannel.write(buffer);
            }

            buffer.clear();
```

```
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

这个应用程序的效果是下面BrasilOpen.txt文件的修改：

```
Brasil Open At Forefront Of Green Movement The Brasil Open, the second
stop of the four-tournament Latin American swing, is held in an area
renowned for its lush natural beauty and stunning beaches. The
tournament has taken a lead in environmental conservation efforts, with
highlights including the planting of 500 trees to neutralise carbon
emissions and providing recyclable materials to local children for use
in craft work.
```

这一套示例应该可以帮助你理解如何去随机访问文件内容.接下来,我们将把SeekableByteChannel接口转型为FileChannel,让我们去访问更多高级的特性.

使用FileChannel

FileChannel在JAVA 4中引入,但是最近它更新了去实现新的SeekableByteChannel接口,结合它们的力量去获取更多的力量.SeekableByteChannel提供了随机访问文件特性,同时FileChannel提供了更高级的特性,比如将文件的一部分直接映射到内存获取更快的访问,以及锁定文件的一部分.

为一个Path获得一个FileChannel可以使用新的 FileChannel.open()方法完成.两个方法都能够去打开或者为给定的Path创建一个文件,然后返回一个新的通道.第一个(最简单的)方法接收文件路径去打开或者创建和一组选项指定文件如何被打开.第二个方法接收文件路径去打开或者创建,一组选项指定文件如何被打开,可选的,一组文件属性被自动设置当文件被创建的时候.

比如,下面的代码使用读/写能力获取指定路径的一个文件通道.

```
Path path = Paths.get("...");
...
try (FileChannel fileChannel = (FileChannel.open(path, EnumSet.of(
                                    StandardOpenOption.READ, StandardOpenOption.WRITE))))
{
...
} catch (IOException ex) {
  System.err.println(ex);
}
```

明确地将一个SeekableByteChannel转型为一个FileChannel对于之前的代码来说是可选的:

```
Path path = Paths.get("...");
...
try (FileChannel fileChannel = (FileChannel)(Files.newByteChannel(path,
                              EnumSet.of(StandardOpenOption.READ,
StandardOpenOption.WRITE))))
{
```

```
…
} catch (IOException ex) {
  System.err.println(ex);
}
```

现在,fileChannel实例可以去访问SeekableByteChannel和FileChannel提供的方法.

直接将一个通道的文件区域映射为内存

FileChannel的主要功能之一就是直接将一个通道的文件的区域映射为内存的能力.这个要多亏了FileChannel.map()方法, 有如下三个参数:

- mode:直接将一个区域映射为内存可用以下三个模式中的一个完成:
MapMode.READ_ONLY(只读映射: 尝试写将抛出ReadOnlyBufferException),
MapMode.READ_WRITE(读/写映射;在产生的缓冲区中的改变可以传播(propagated)到文件中,并对于映射同一文件的其它程序可见),或者MapMode.PRIVATE(copy-on-write映射;在产生的缓冲区中的改变不能传播到文件中,对于其它程序不可见).
- position:映射区域在文件中从指定的位置开始(非负数).
- size:表明映射区域的大小(0 ≤ size ≤ Integer.MAX_VALUE).

注意: 只有用来读取的打开的通道可被映射为read-only, 只有用来读和写的打开的通道可以被映射为read/write或者private.

---

map()方法将返回一个MappedByteBuffer,实际上代表了提取的区域.这个类继继承了ByteBuffer,有以下三个方法,更多的细节你可以在官方文档
http://download.oracle.com/javase/7/docs/api/index.html 中找到.

- force(): 强制将改变了的缓冲区传播到原始文件
- load(): 加载缓冲区的内容到物理内存
- isLoaded(): 验证缓冲区内容是否在物理内存中

接下来的应用程序为文件BrasilOpen.txt获得一个新的通道(位于C:\rafaelnadal\tournaments\2009),然后将它的全部内容映射使用READ_ONLY模式映射到一个字节缓冲区.为了测试操作完成成功,下面是一个字节缓冲区内容的输出:

```
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/tournaments/2009", "BrasilOpen.txt");
        MappedByteBuffer buffer = null;

        try (FileChannel fileChannel = (FileChannel.open(path,
EnumSet.of(StandardOpenOption.READ)))) {
            buffer = fileChannel.map(FileChannel.MapMode.READ_ONLY, 0, fileChannel.size());

        } catch (IOException ex) {
            System.err.println(ex);
        }

        if (buffer != null) {
            try {
```

```
                Charset charset = Charset.defaultCharset();
                CharsetDecoder decoder = charset.newDecoder();
                CharBuffer charBuffer = decoder.decode(buffer);
                String content = charBuffer.toString();
                System.out.println(content);

                buffer.clear();
            } catch (CharacterCodingException ex) {
                System.err.println(ex);
            }
        }
    }
}
```

如果一切工作正常,你应该看到BrasilOpen.txt的内容输出到屏幕上.

锁定文件通道

文件锁是一个限制范文一个文件或者数据的其它块来保证两个或者更多用户不能同时修改同样的文件的机制.这防止了经典的调解更新场景(FIMXE: This prevent the classic interceding update scenario).通常文件被锁定当第一个用户访问它然后保持锁定(可以读,但是不能修改知道用户结束了).

文件锁定的明确行为是平台依赖的.在一些平台,文件锁是可咨询(advisory)的(任何应用程序可以访问文件如果应用程序没有检查文件锁),而对于其它的是强制的(文件锁阻止任何应用程序访问一个文件).

我们可以在Java应用程序中通过NIO API使用文件锁.然后,不能保证文件锁机制将一直如你预期的工作.底层的操作系统是否支持,有时候,一个错误的实现可能影响预期的行为.记住下面的:

- "文件锁定以整个 Java 虚拟机来保持.但它们不适用于控制同一虚拟机内多个线程对文件的访问."(Java平台SE 7 官方文档, http://download.oracle.com/javase/7/docs/api/java/nio/channels/FileLock.html).
- Windows为你负责锁定目录和其它的结构,所以一个删除,重命名,或者写操作将失败,如果另一个进程打开文件.因此, 在一个系统上创建一个Java锁将失败.
- Linux内核管理一组功能,称为咨询锁定机制(FIXME: advisory locking mechanisms).另外,你可以通过强制锁在内核级别上锁定.因此,当使用Java锁,记住这个方面.

FileChannel类为文件锁提供了四种方法:两个lock()方法和两个tryLock()方法.lock()方法阻塞应用程序直到期望的锁可以被重新取回,而tryLock()方法不会阻塞应用程序,返回null或者抛出异常如果文件已经被锁定.有一个lock()/tryLock()方法用来重新取回在这个通道的文件上的一个独占锁和在通道的文件上的一个区域重新取回一个锁-这个方法允许一个锁是共享的.

为了证明文件锁定,我们将看两个应用程序.第一个锁定一个名为vamos.txt的文件(位于C:\rafaelnadal\email)两分钟,同时往里面写入一些文本.第二个应用程序将尝试去在相同的文件进行写入.如果文件被成功锁定2分钟.然后第二个应用程序将抛出java.io.IOException,然后输出像下面的信息:

```
The process cannot access the file because another process has locked
a portion of the file.
```

这里是第一个应用程序：

```
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/email", "vamos.txt");
        ByteBuffer buffer = ByteBuffer.wrap("Vamos Rafa!".getBytes());

        try (FileChannel fileChannel = (FileChannel.open(path,
EnumSet.of(StandardOpenOption.READ, StandardOpenOption.WRITE)))) {

            // Use the file channel to create a lock on the file.
            // This method blocks until it can retrieve the lock.
            FileLock lock = fileChannel.lock();

            // Try acquiring the lock without blocking. This method returns
            // null or throws an exception if the file is already locked.
            //try {
            //    lock = fileChannel.tryLock();
            //} catch (OverlappingFileLockException e) {
            // File is already locked in this thread or virtual machine
            //}

            if (lock.isValid()) {

                System.out.println("Writing to a locked file ...");
                try {
                    Thread.sleep(60000);
                } catch (InterruptedException ex) {
                    System.err.println(ex);
                }
                fileChannel.position(0);
                fileChannel.write(buffer);
                try {
                    Thread.sleep(60000);
                } catch (InterruptedException ex) {
                    System.err.println(ex);
                }
            }

            // Release the lock
            lock.release();
            System.out.println("\nLock released!");
        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

运行之前的应用程序，然后在2分钟中内，并行启动下面的应用程序：

```
public class Main {
    public static void main(String[] args) {
        Path path = Paths.get("C:/rafaelnadal/email", "vamos.txt");
        ByteBuffer buffer = ByteBuffer.wrap("Hai Hanescu !".getBytes());

        try (FileChannel fileChannel = (FileChannel.open(path,
```

```
EnumSet.of(StandardOpenOption.READ, StandardOpenOption.WRITE))) {

            fileChannel.position(0);
            fileChannel.write(buffer);

        } catch (IOException ex) {
            System.err.println(ex);
        }
    }
}
```

你应该可以看到第二个应用程序只有在2分钟之后， 锁被释放后才能写入vamos.txt.

使用FileChannel拷贝文件

FileChannel提供了一种新的方式去拷贝文件.你可以通过一个直接的或者非直接的ByteBuffer使用FileChanel,使用FileChannel.transferTo()或者FileChannel.transferFrom(),或者使用FileChannel.map().

使用FileChannel和一个直接的或者非直接的ByteBuffer拷贝文件

为了使用FileChannel和一个直接或者非直接的ByteBuffer拷贝文件,我们需要一个通道用作源文件,一个通道用作目标文件,和一个直接的或非直接的ByteBuffer.比如,下面的代码片段将使用4KB大小的直接缓冲区拷贝文件 Rafa Best Shots.mp4(位于C:\rafaelnadal\tournaments\2009\videos directory)到 C:\ root using a direct ByteBuffer.

```
…
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");
int bufferSizeKB = 4;
int bufferSize = bufferSizeKB * 1024;
…
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))))
{
     // 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();
     }
} catch (IOException ex) {
```

```
  System.err.println(ex);
}
…
```

为了使用一个非直接缓冲区,只要使用下面的行:

```
ByteBuffer bytebuffer = ByteBuffer.allocate(bufferSize);
```

替换行:

```
ByteBuffer bytebuffer = ByteBuffer.allocateDirect(bufferSize);
```

使用FileChannel.transferTo()或者FileChannel.transferFrom()拷贝文件

FileChannel.transferTo()从一个通道文件传输字节到给定的可写的字节通道.你选择位置,要传输的最大数目的字节,和目标通道,FileChannel.transferTo()返回传输的字节数目.下面的示例传输Rafa Best Shots.mp4的整个内容.

```
…
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.WRITE))))
{
            fileChannel_from.transferTo(0L, fileChannel_from.size(), fileChannel_to);
} catch (IOException ex) {
  System.err.println(ex);
}
…
```

或者,你可以使用FileChannel.transferFrom()从一个给定的可读字节通道传输字节到这个通道的文件.为了这样,修改之前的代码通过使用下面的行:

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

替换行:

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

使用FileChannel.map()拷贝

在之前的章节中你看到如何使用MappedByteBuffer映射通道的文件的一个区域到内存中.在这节,我们扩展(FIXME: extrapolate)那个实例去拷贝Rafa Best Shots.mp4 的内容:

```
…
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))))
{
     MappedByteBuffer buffer = fileChannel_from.map(FileChannel.MapMode.READ_ONLY, 0,

 fileChannel_from.size());
```

```
    fileChannel_to.write(buffer);
    buffer.clear();
} catch (IOException ex) {
  System.err.println(ex);
}
…
```

基准测试FileChannel的复制性能

在前面的三节你看到了3种不同的方式去使用FileChannel拷贝一个文件.Java也提供了另外一组方案用来拷贝一个文件,包括使用Files.copy()方法或者缓冲的/非缓冲的流和字节数组.你应该选择哪一个?这是一个困难的问题,它的答案依赖于许多因素.这个关注于一个因此,速度,因为快速完成一个拷贝任务会提高生产率,在一些情况中,是成功的关键.因此,这节实现了一个应用程序,比较以下的每个解决方案每次拷贝花费多长时间.

- FileChannel和非直接缓冲区
- FileChannel和直接缓冲区
- FileChannel.transferTo()
- FileChannel.transferFrom()
- FileChannel.map()
- 使用缓冲的流和字节数组
- 使用非缓冲的流和字节数组
- Files.copy() (Path到Path, InputStram到Path, 和Path到OutputStream)

测试基于以下条件:

- 拷贝文件类型:MP4 video(文件名为Rafa Best Shots.mp4,它一开始位于C:\rafaelnadal\tournaments\2009\videos)
- 拷贝文件大小:58.3MB
- 缓冲大小测试:4KB, 16KB, 32KB, 64KB, 128KB, 256KB, 和 1024KB
- 机器: Mobile AMD Sempron Processor 3400 + 1.80 GHz, 1.00GB RAM, 32-bit OS, Windows 7 Ultimate
- 测量(Measurement)类型:使用System.nanoTime()方法
- ;头三个运行被忽视去获得一个趋势;第一次运行总是比随后的运行慢.

接下来列出的应用程序在源代码中是可用的,在Apress.com从这本书的页面的这节下载.

```
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;
        int bufferSize = bufferSizeKB * 1024;

        deleteCopied(copy_to);

        //FileChannel and indirect 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 an 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 an 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 and 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.WRITE)))) {

            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 and transfer from
        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(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 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) + "
seconds");
        } 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);
        }
    }
}
```

这个应用程序的输出难以整理,因为涉及许多数字.所以我绘制了一些数据去给你一个几种比较的结果的清晰的图片.在下面的章节的图中展示.Y轴在这些图中是估算(estimated)的时间以秒的方

式表达,X轴是使用的缓冲区的大小(或者运行数目,在跳过头三个运行后)

FileChannel和非直接缓冲区 VS FileChannel和直接缓冲区

如图7-11所示,它看起来缓冲区小于256KB,非直接缓冲区更快,同时缓冲区大于256KB,直接缓冲区稍微快写(见图7-11).

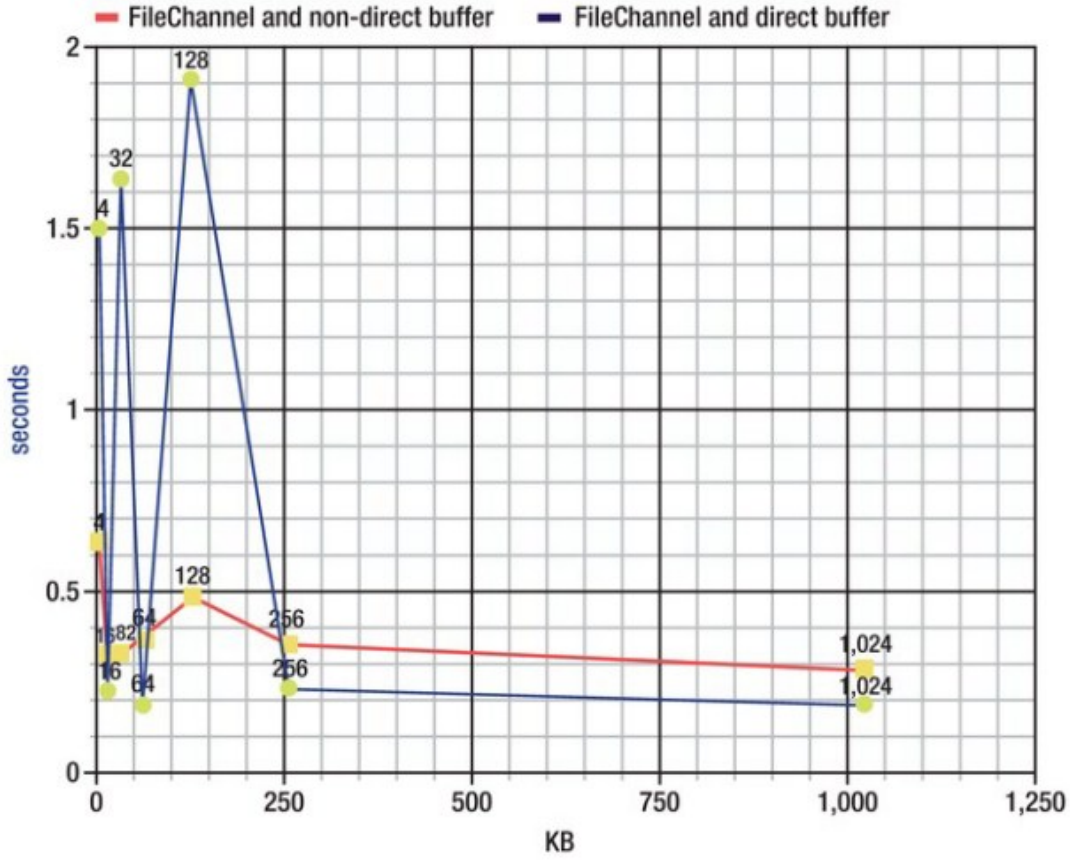

Figure 7-11. *FileChannel and non-direct buffer vs. FileChannel and direct buffer*

FileChannle.transferTo() VS FileChannel.transferFrom() VS FileChannel.map()

如图7-12所示,看上去通过7次连续运行transferTo()和transferFrom()几乎是同样的,同时FileChannel.map()是最慢的解决方案.

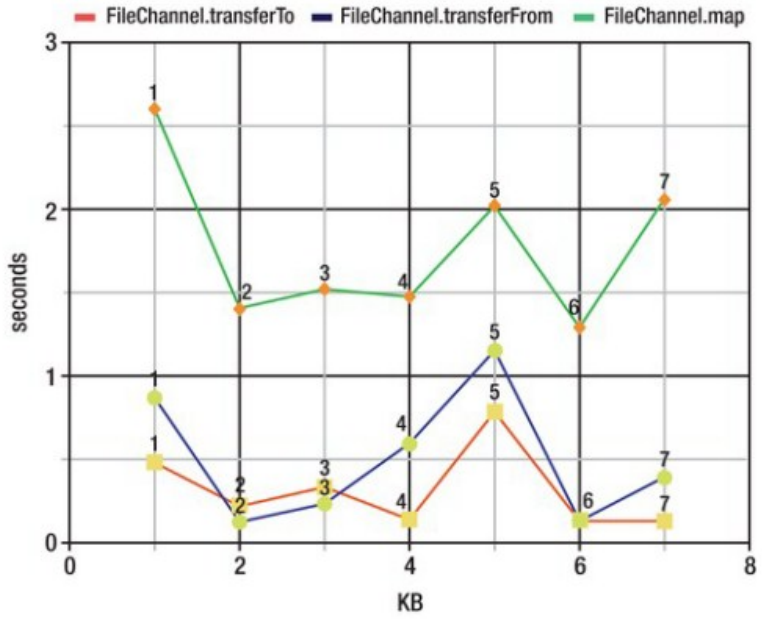

Figure 7-12. *FileChannel.transferTo() vs. FileChannel.transferFrom() vs. FileChannel.map()*

三种不同的`Files.copy()`方法

如图7-13所示,最快的`Files.copy()`方法是`Path`到`Path`,接下来是`Path`到`OutputStream`,最后是`InputStream`到`Path`.

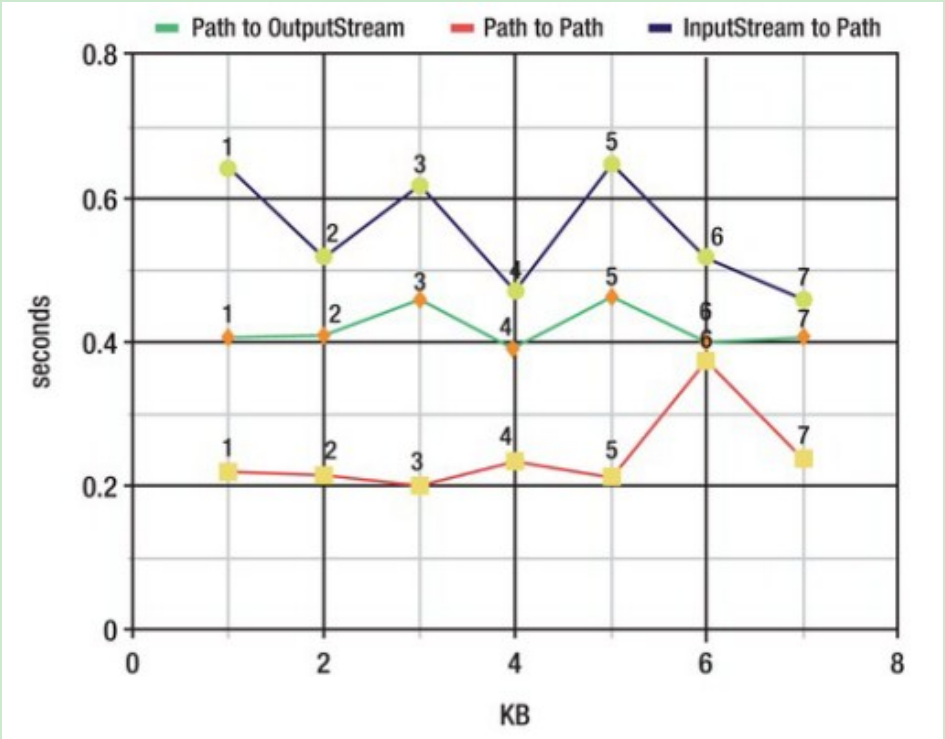

Figure 7-13. *Files.copy() approches*

`FileChannel`和非直接缓冲区 VS `FileChannel.transferTo()` VS `Path`到`Path`

作为最终的测试,我们从上面的三个图解中选择最快的结果,将它们一起放在图7-14.因此我们不

为FileChannel.transferTo()和Path到Path指定缓冲区,我们通过7次运行参考平均时间.如你所看到的,Path到Path的Files.copy()看上去是拷贝一个文件的最快的方案.

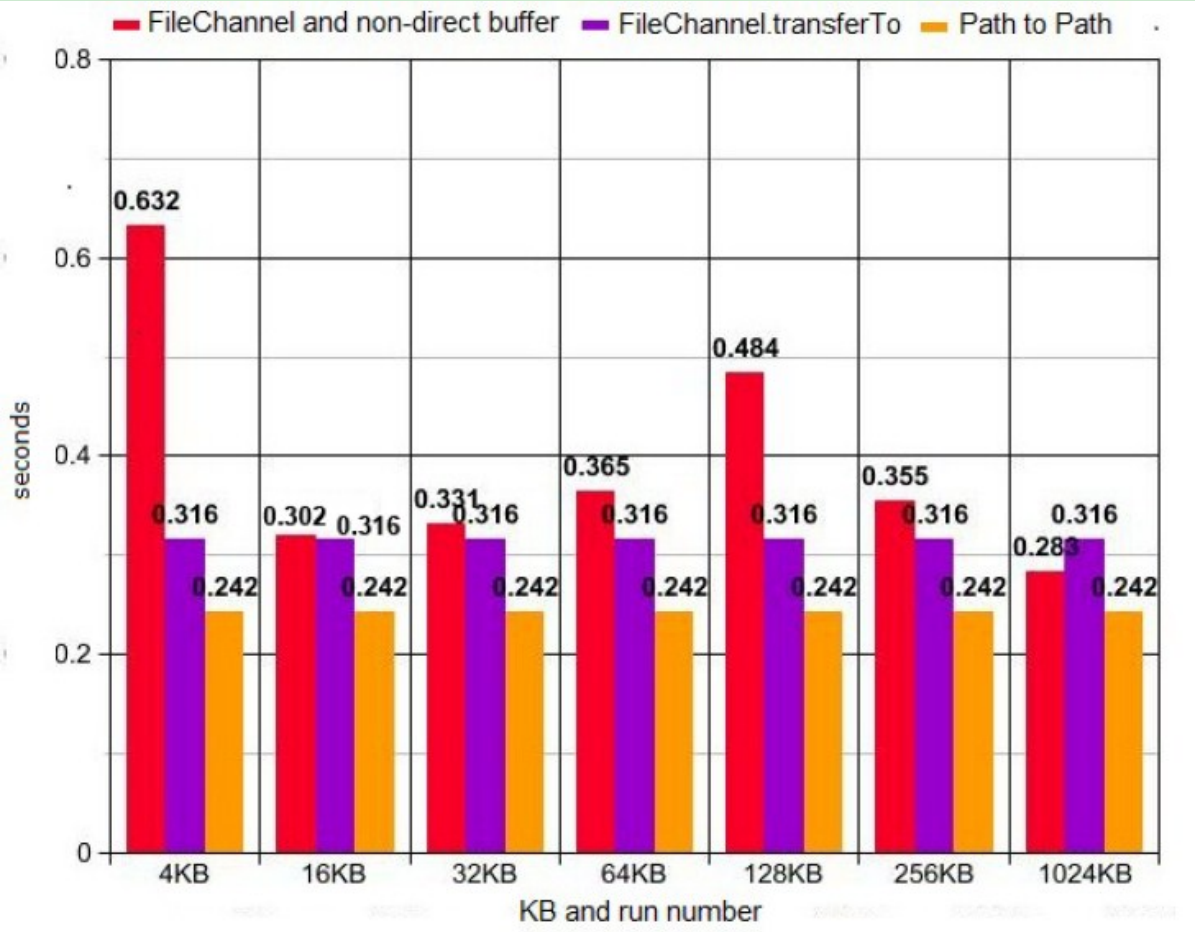

Figure 7-14. FileChannel with non-direct buffer vs. FileChannel.transferTo() vs. Path to Path

总结

这章以ByteBuffer类的一个简短的概览开始.这个类通常和SeekableByteChannel和FileChannel使用.接下来使用应用程序详细化SeekableByteChannel接口,应用程序将随机地读和写文件去完成不同类型的通用的任务.然后你看到使用RAF的能力如何获得一个FileChannel和探索FileChannel提供的主要功能,包括直接映射文件的一部分为内存来更快的访问,锁定文件的一部分,和从一个绝对位置读和写字节而不是影响通道当前的位置.这章以一个基准测试应用程序结束,尝试去决定一个最快的文件拷贝方式,通过比较FileChannel性能针对其他的通用的方法,比如Files.copy(),使用缓冲的流和字节数组,和使用非缓冲的流和字节数组.