(randomAccessFile.jsp)Previous

Java NIO - Channels and Memory mapping

Quiz (http://www.studytrails.com/quiz/java-io/Channels_And_Memory_Mapping/1)

Next (pipes-print-writer.jsp)

Channel Classes

- *Channel*- A channel represents a connection to a file, socket or any other component that can perform IO operations. A channel reads or writes data to a byte buffer. Channels are generally thread safe. A channel may be specified to be read only or both readable and writable.
- *ReadableByteChannel* - A ReadableByteChannel permits read operation on a buffer, allowing only one thread to read at a time.
- *WritableByteChannel* - A WritableByteChannel permits write operation on a buffer, allowing only one thread to read at a time.
- *ByteChannel* - A ByteChannel extends both ReadableByteChannel and WritableByteChannel and allows both read and write.
- *SeekableByteChannel* - A SeekableByteChannel extends ByteChannel and allows to maintain and modify the current position on the underlying entity to which it is connected. It has methods to get the size of the underlying entity or truncate it to a given size, if permitted.
- *GatheringByteChannel*-A GatheringByteChannel is used to write data from multiple buffers into a single channel.
- *ScatteringByteChannel*-A ScatteringByteChannel is used to read data from a channel into multiple buffers.

File Channel

A FileChannel is used to read and write data to a file. It implements seekableByteChannel, ScatteringByteChannel and GatheringByteChannel. It is possible to map a region of file directly into memory. Data can be transferred to another channel or from another channel using the transferTo(..) and transferFrom(..) methods. These methods use the underlying optimization of the operating system. File locking can be applied to manage access between multiple processes. The methods are thread safe and a thread that wishes to modify the position may be blocked until another thread is acting upon the file.

Scatter Read using File Channel

Data from a File Channel can be read into multiple buffers. This is known as a scatter read. Here's and example demonstrating a scatter read.

```
1  // create a temp file
2  Path tempFile =
3  Files.createTempFile("fileChannelExample", "txt",new FileAttribute<?>[0]);
4
5  // write some data to this file
6  PrintWriter writer = new PrintWriter(tempFile.toFile());
7  writer.println("First Line");
8  writer.println("Second Line");
```

```
writer.println("Third Line");
writer.close();
// get an input stream for this file.
FileInputStream in = new FileInputStream(tempFile.toFile());

// get the fileChannel for this input stream
FileChannel fileChannel = in.getChannel();

// get the position
System.out.println(fileChannel.position());//print 0

// create a char buffer
ByteBuffer buffer = ByteBuffer.allocate(11);
// read the fist line - 10 characters into the byte buffer
fileChannel.read(buffer);
// get the position of the file
System.out.println(fileChannel.position());// prints 11
// we have read the first 10 chars into the buffer now. find out the
// total size of the file
System.out.println(fileChannel.size()); // prints 37
// we read the next 27 characters in two buffers using a scattering read
ByteBuffer buffer1 = ByteBuffer.allocate(14);
ByteBuffer buffer2 = ByteBuffer.allocate(12);
fileChannel.read(new ByteBuffer[] { buffer1, buffer2 });
// lets see the contents of the buffers
buffer1.flip();
buffer2.flip();
System.out.print(Charset.defaultCharset().decode(buffer1)); // prints "Second
System.out.print(Charset.defaultCharset().decode(buffer2));// prints "Third L
fileChannel.close();
```

Scatter Write using File Channel

we can write the bytes back to the file using a scattering write. The filechannel was created on the inputstream so the channel is only readable but not writable. we create a filechannel from an output stream

```
FileOutputStream out = new FileOutputStream(tempFile.toFile());
        FileChannel fileOutputChannel = out.getChannel();
        ByteBuffer buffer3 = Charset.defaultCharset().encode(
                CharBuffer.wrap("Line1\n".toCharArray()));
        ByteBuffer buffer4 = Charset.defaultCharset().encode(
                CharBuffer.wrap("Line2".toCharArray()));
        fileOutputChannel.write(new ByteBuffer[] { buffer3, buffer4 });
        // we force the update to the file
        fileOutputChannel.force(true);
        // lets look at the position in the file
        System.out.println(fileOutputChannel.position());
```

Reading Huge Files using Memory Mapped Buffer

Java NIO allows reading huge files directly from the memory. i.e. The file need not be loaded into the JVM. All reads and writes on the byte buffer happen direclty on the file.

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

public class ReadingHugeFilesUsingMemoryMappedBuffer {
    /**
     * use a MappedByteBuffer to wrap a huge file. Using a MappedByteBuffer c
```

```
10          * not load the file in JVM but reads it directly off the file system
11          * memory. The file can be opened in read, write or private mode.
12          */
13         // to test you can use any video movie file if you dont have any other la
14         // file for testing.
15         private static String hugeFile = "A Huge File";
16
17         public static void main(String[] args) throws IOException {
18             File file = new File(hugeFile);
19             FileChannel fileChannel = new RandomAccessFile(file, "r").getChannel(
20             MappedByteBuffer buffer = fileChannel.map(
21                     FileChannel.MapMode.READ_ONLY, 0, fileChannel.size());
22             // the buffer now reads the file as if it were loaded in memory. note
23             // that for smaller files it would be faster
24             // to just load the file in memory
25             // lets see if this buffer is loaded fully into memory
26             System.out.println(buffer.isLoaded());
27             // the mappedbytebuffer can be used as a normal buffer to do read and
28             // write operations
29             // read the size
30             System.out.println(buffer.capacity());
31
32
33         }
34     }
```

(randomAccessFile.jsp)Previous

Java NIO - Channels and Memory mapping

Quiz (http://www.studytrails.com/quiz/java-io/Channels_And_Memory_Mapping/1)

Next (pipes-print-writer.jsp)

Sitemap (http://www.studytrails.com/sitemap_index.xml)