Processing large files efficiently in Java – part 1

Posted on March 4, 2015 by Arulkumaran Kumaraswamipillai

Q1. What are the key considerations in processing large files?
A1. Before jumping into coding, get the requirements.

#1. Processing a file involves reading from the disk, processing, and writing back to the disk. It is also a trade off in terms of what is more important to you like having better I/O, better CPU usage, and better memory usage. It is important to conduct profiling to monitor CPU usage, memory usage, and I/O efficiency.

1) Reading the data from the disk can be **I/O-heavy**,
2) Storing the read data in the Java heap can be **memory-heavy**,

3) Parsing the data can be **CPU-heavy**.
4) Writing the processed data back to the disk can be **I/O-heavy**.

#2. File types to process. Only ASCII, only binary, or both ASCII and binary. If you need to handle only ASCII files, you could write a simple bash script that divide files into smaller pieces and read them as usual.

```
1 split -b  5M my-huge-file.csv segment
```

But if you have a requirement to handle binary formats as well, then splitting files approach won't work. If it is an XML file, then favor using an XML file parser like StAX (**St**reaming **A**PI for **X**ML).

#3. In order to better utilize I/O, CPU, and memory, you will have to read, parse, and write in chunks. You need to process regions of data incrementally using **memory mapped files**. The good thing about the memory mapped files is that they do not consume virtual memory or paging space since it is backed by file data on disk. But, you can get OutOfMemory errors for very large files.

#4. You can also introduce **multi-threading** with a pool of finite number of threads to improve CPU and I/O efficiency at the cost of memory. For example:

"You can create a big byte buffer, run several threads that read bytes from a file into that buffer in parallel, when ready find first end of line, make a string object, find next, and repeat these sequences.

Again, it is a trade-off, and profile and tune your application in terms of thread pool size, allocated heap memory, garbage collection algorithms, etc.

Q2. What are the different ways to read a file in Java?
A2. There are many ways. Here are some examples and timings using a **5MB** file, **250MB** file, and a **1GB** file.

As a Java Architect

Java architecture & design concepts interview Q&As with diagrams | What should be a typical Java EE architecture?

1. Read a 5MB file line by line with a scanner class

Total elapsed time: **271 ms**. Light on memory usage, but heavy on I/O.

```
1  package com.large.file;
2
3  import java.io.File;
4  import java.io.FileNotFoundException;
5  import java.util.Scanner;
6  import java.util.concurrent.TimeUnit;
7
8  public class ScannerRead {
9
10     /**
11      * This solution will iterate through all th
12      * without keeping references to them  i.e. 
13      */
14
15     public static void main(String[] args) {
16
17         long startTime = System.nanoTime();
18         try (Scanner sc = new Scanner(new File("
19             long freeMemoryBefore = Runtime.getR
20             while (sc.hasNextLine()) {
21
22                 String line = sc.nextLine();
23                 // parse line
24                 //System.out.println(line);
25             }
26
27             // note that Scanner suppresses exce
28             if (sc.ioException() != null) {
29                 sc.ioException().printStackTrace
30             }
31         } catch (FileNotFoundException e) {
32             e.printStackTrace();
33         }
34
35         long endTime = System.nanoTime();
36         long elapsedTimeInMillis = TimeUnit.MILL
37         System.out.println("Total elapsed time:
38     }
39 }
40
```

2. Reading a 5MB file with Java NIO using memory mapped files

Total elapsed time: **43 ms**. Efficient on I/O. More work is required to process the buffer.

```
1  package com.large.file;
2
3  import java.io.IOException;
4  import java.io.RandomAccessFile;
5  import java.nio.CharBuffer;
6  import java.nio.MappedByteBuffer;
7  import java.nio.channels.FileChannel;
8  import java.nio.charset.StandardCharsets;
9  import java.util.concurrent.TimeUnit;
10
11 public class ByteBufferRead {
12
13     public static void main(String[] args) {
14
15         long startTime = System.nanoTime();
16
17         try {
18             RandomAccessFile aFile = new RandomA
19             FileChannel inChannel = aFile.getCha
20             MappedByteBuffer buffer = inChannel.
21
22             buffer.load();
23
24             CharBuffer charBuffer = StandardChar
25             String read = charBuffer.toString();
26
27             //System.out.println(read);
28
29             buffer.clear(); // do something with
30             inChannel.close();
31             aFile.close();
32
33         } catch (IOException ioe) {
34             ioe.printStackTrace();
35         }
36
37         long endTime = System.nanoTime();
38         long elapsedTimeInMillis = TimeUnit.MILL
39         System.out.println("Total elapsed time:
40     }
41 }
42
```

3. Reading a 5MB file line by line with Java 8 Stream

Total elapsed time: **160 ms**.

```
1  package com.large.file;
2
3  import java.io.IOException;
4  import java.nio.charset.StandardCharsets;
5  import java.nio.file.Files;
6  import java.nio.file.Path;
7  import java.nio.file.Paths;
8  import java.util.concurrent.TimeUnit;
9  import java.util.stream.Stream;
10
11 public class Java8StreamRead {
```

```
12
13      public static void main(String[] args) {
14
15          long startTime = System.nanoTime();
16          Path file = Paths.get("c:/temp/my-large-
17          try
18          {
19              //Java 8: Stream class
20              Stream<String> lines = Files.lines(
21
22              for( String line : (Iterable<String>
23              {
24                 //System.out.println(line);
25              }
26
27          } catch (IOException ioe){
28              ioe.printStackTrace();
29          }
30
31          long endTime = System.nanoTime();
32          long elapsedTimeInMillis = TimeUnit.MILL
33          System.out.println("Total elapsed time:
34      }
35  }
```

Reading a 250.0MB file

Scanner approach: Total elapsed time: **7062 ms**
Maped Byte Buffer: Total elpased time: **1220 ms**
Java 8 Stream: Total elapsed time: **1024 ms**

Reading a 1.0GB file

Sanner approach: Total elapsed time: **15627 ms**
Maped Byte Buffer: Exception in thread “main”
java.lang.**OutOfMemoryError**: Java heap space
Java 8 Stream: Total elapsed time: **3124 ms**

Java 8 Rocks. Monitor your application to see if it is more I/O bound, memory bound, or CPU bound. In the next post will introduce file parsing and multi-threading to improve efficiency.

Popular Posts

♦ 11 Spring boot interview questions & answers

884 views

♦ Q11-Q23: Top 50+ Core on Java OOP Interview Questions & Answers

838 views

18 Java scenarios based interview Questions and Answers

454 views

001A: ♦ 7+ Java integration styles & patterns interview questions & answers

410 views

♦ 7 Java debugging interview questions & answers

314 views

♦ 10 ERD (Entity-Relationship Diagrams) Interview Questions and Answers

308 views

01b: ♦ 13 Spring basics Q8 – Q13 interview questions & answers

298 views

01: ♦ 15 Ice breaker questions asked 90% of the time in Java job interviews with hints

286 views

♦ Q24-Q36: Top 50+ Core on Java classes, interfaces and generics interview questions & answers

265 views

8 Git Source control system interview questions & answers

221 views

Bio | Latest Posts

Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

About Arulkumaran Kumaraswamipillai

Mechanical Eng to freelance Java developer in 3 yrs. Contracting since 2003, and attended 150+ Java job interviews, and often got 4 - 7 job offers to choose from. It pays to prepare. So, published Java interview Q&A books via Amazon.com in 2005, and sold 35,000+ copies. Books are outdated and replaced with this subscription based site.

‹ How to become a self-taught Java (or any) developer everyone wants to hire?

Processing large files efficiently in Java – part 2 ›

Posted in IO, Memory Management, Performance

Empowers you to open more doors, and fast-track

Technical Know Hows

☀ Java generics in no time ☀ Top 6 tips to transforming your thinking from OOP to FP ☀ How does a HashMap internally work? What is a hashing function?
☀ 10+ Java String class interview Q&As ☀ Java auto un/boxing benefits & caveats ☀ Top 11 slacknesses that can come back and bite you as an experienced Java developer or architect

Non-Technical Know Hows

☀ 6 Aspects that can motivate you to fast-track your career & go places ☀ Are you reinventing yourself as a Java developer? ☀ 8 tips to safeguard your Java career against offshoring ☀ My top 5 career mistakes

Prepare to succeed

☀ Turn readers of your Java CV go from “Blah blah” to “Wow”? ☀ How to prepare for Java job interviews? ☀ 16 Technical Key Areas ☀ How to choose from multiple Java job offers?

Select Category ▼

© Disclaimer

 ↑