纸上得来终觉浅 绝知此事要躬行——https://github.com/sgq0085/learn

高效读取大数据文本文件（上亿行数据）

博客分类：
- Java基础-流与文件

LineNumberReaderRandomAccessFileBufferedRandomAccessFile文本读取大数据

一.前言

本文是对大数据文本文件读取（按行读取）的优化，目前常规的方案（限于JDK）有三种，第一种LineNumberReader，第二种RandomAccessFile，第三种是内存映射文件（详见http://sgq0085.iteye.com/blog/1318622）在RandomAccessFile基础上调用getChannel().map(...)。

1.LineNumberReader

按行读取，只能从第一行向后遍历，到需要读取的行时开始读入，直到完成；在我的测试用例中，读取1000W行数据每次5万行，用时93秒，效率实测比RandomAccessFile要高，但读取一亿跳数据时效率太低了（因为每次都要从头遍历），因为测试时超过1个小时，放弃测试；

2.RandomAccessFile

实际不适用于这种大数据读取，RandomAccessFile是为了磁盘文件的随机访问，所以效率很低，1000w行测试时用时140秒，一亿行数据测试用时1438秒但由于可以通过getFilePointer方法记录位置，并通过seek方法指定读取位置，所以从理论上比较适用这种大数据按行读取的场景；

RandomAccessFile只能按照8859_1这种方法读取，所以需要对内容重新编码，方法如下

Java代码

```
1. new String(pin.getBytes("8859_1"), "")
```

3.内存映射文件

由于每行数据大小不同，内存映射文件在这种情况下不适用，其他情况请参考我的博客（详见http://sgq0085.iteye.com/blog/1318622）

二.解决方案

如果在RandomAccessFile基础上，整合内部缓冲区，效率会有提高，测试过程中1000w行数据用时1秒，1亿行数据用时103（比1438秒快了13倍左右）

BufferedRandomAccessFile

网上已经有实现，代码如下：

Java代码

```
1. package com.gqshao.file.io;
2.
3. import java.io.File;
4. import java.io.FileNotFoundException;
5. import java.io.IOException;
6. import java.io.RandomAccessFile;
7. import java.util.Arrays;
8.
9. public class BufferedRandomAccessFile extends RandomAccessFile {
10.     static final int LogBuffSz_ = 16; // 64K buffer
11.     public static final int BuffSz_ = (1 << LogBuffSz_);
12.     static final long BuffMask_ = ~(((long) BuffSz_) - 1L);
13.
14.     private String path_;
15.
16.     /*
17.      * This implementation is based on the buffer implementation in Modula-3's
18.      * "Rd", "Wr", "RdClass", and "WrClass" interfaces.
19.      */
20.     private boolean dirty_; // true iff unflushed bytes exist
21.     private boolean syncNeeded_; // dirty_ can be cleared by e.g. seek, so track sync separately
22.     private long curr_; // current position in file
23.     private long lo_, hi_; // bounds on characters in "buff"
24.     private byte[] buff_; // local buffer
25.     private long maxHi_; // this.lo + this.buff.length
26.     private boolean hitEOF_; // buffer contains last file block?
27.     private long diskPos_; // disk position
28.
29.      /*
30.      * To describe the above fields, we introduce the following abstractions for
31.      * the file "f":
32.      *
33.      * len(f) the length of the file curr(f) the current position in the file
34.      * c(f) the abstract contents of the file disk(f) the contents of f's
35.      * backing disk file closed(f) true iff the file is closed
36.      *
37.      * "curr(f)" is an index in the closed interval [0, len(f)]. "c(f)" is a
38.      * character sequence of length "len(f)". "c(f)" and "disk(f)" may differ if
39.      * "c(f)" contains unflushed writes not reflected in "disk(f)". The flush
40.      * operation has the effect of making "disk(f)" identical to "c(f)".
41.      *
42.      * A file is said to be *valid* if the following conditions hold:
43.      *
44.      * V1. The "closed" and "curr" fields are correct:
45.      *
46.      * f.closed == closed(f) f.curr == curr(f)
```

```
     *
     * V2. The current position is either contained in the buffer, or just past
     * the buffer:
     *
     * f.lo <= f.curr <= f.hi
     *
     * V3. Any (possibly) unflushed characters are stored in "f.buff":
     *
     * (forall i in [f.lo, f.curr): c(f)[i] == f.buff[i - f.lo])
     *
     * V4. For all characters not covered by V3, c(f) and disk(f) agree:
     *
     * (forall i in [f.lo, len(f)): i not in [f.lo, f.curr) => c(f)[i] ==
     * disk(f)[i])
     *
     * V5. "f.dirty" is true iff the buffer contains bytes that should be
     * flushed to the file; by V3 and V4, only part of the buffer can be dirty.
     *
     * f.dirty == (exists i in [f.lo, f.curr): c(f)[i] != f.buff[i - f.lo])
     *
     * V6. this.maxHi == this.lo + this.buff.length
     *
     * Note that "f.buff" can be "null" in a valid file, since the range of
     * characters in V3 is empty when "f.lo == f.curr".
     *
     * A file is said to be *ready* if the buffer contains the current position,
     * i.e., when:
     *
     * R1. !f.closed && f.buff != null && f.lo <= f.curr && f.curr < f.hi
     *
     * When a file is ready, reading or writing a single byte can be performed
     * by reading or writing the in-memory buffer without performing a disk
     * operation.
     */

    /**
     * Open a new <code>BufferedRandomAccessFile</code> on <code>file</code>
     * in mode <code>mode</code>, which should be "r" for reading only, or
     * "rw" for reading and writing.
     */
    public BufferedRandomAccessFile(File file, String mode) throws IOException {
        this(file, mode, 0);
    }

    public BufferedRandomAccessFile(File file, String mode, int size) throws IOException {
        super(file, mode);
        path_ = file.getAbsolutePath();
        this.init(size);
    }

    /**
     * Open a new <code>BufferedRandomAccessFile</code> on the file named
     * <code>name</code> in mode <code>mode</code>, which should be "r" for
     * reading only, or "rw" for reading and writing.
     */
    public BufferedRandomAccessFile(String name, String mode) throws IOException {
        this(name, mode, 0);
    }

    public BufferedRandomAccessFile(String name, String mode, int size) throws FileNotFoundException {
        super(name, mode);
        path_ = name;
        this.init(size);
    }

    private void init(int size) {
        this.dirty_ = false;
        this.lo_ = this.curr_ = this.hi_ = 0;
        this.buff_ = (size > BuffSz_) ? new byte[size] : new byte[BuffSz_];
        this.maxHi_ = (long) BuffSz_;
        this.hitEOF_ = false;
        this.diskPos_ = 0L;
    }

    public String getPath() {
        return path_;
    }

    public void sync() throws IOException {
        if (syncNeeded_) {
            flush();
            getChannel().force(true);
            syncNeeded_ = false;
        }
    }

//      public boolean isEOF() throws IOException
//      {
//          assert getFilePointer() <= length();
//          return getFilePointer() == length();
//      }

    public void close() throws IOException {
        this.flush();
        this.buff_ = null;
        super.close();
    }

    /**
     * Flush any bytes in the file's buffer that have not yet been written to
     * disk. If the file was created read-only, this method is a no-op.
     */
    public void flush() throws IOException {
        this.flushBuffer();
    }

    /* Flush any dirty bytes in the buffer to disk. */
    private void flushBuffer() throws IOException {
        if (this.dirty_) {
            if (this.diskPos_ != this.lo_)
                super.seek(this.lo_);
            int len = (int) (this.curr_ - this.lo_);
            super.write(this.buff_, 0, len);
            this.diskPos_ = this.curr_;
            this.dirty_ = false;
        }
    }

    /*
     * Read at most "this.buff.length" bytes into "this.buff", returning the
     * number of bytes read. If the return result is less than
     * "this.buff.length", then EOF was read.
     */
    private int fillBuffer() throws IOException {
        int cnt = 0;
        int rem = this.buff_.length;
        while (rem > 0) {
            int n = super.read(this.buff_, cnt, rem);
            if (n < 0)
                break;
            cnt += n;
            rem -= n;
        }
        if ((cnt < 0) && (this.hitEOF_ = (cnt < this.buff_.length))) {
            // make sure buffer that wasn't read is initialized with -1
```

```
        Arrays.fill(this.buff_, cnt, this.buff_.length, (byte) 0xff);
    }
    this.diskPos_ += cnt;
    return cnt;
}

/*
 * This method positions <code>this.curr</code> at position <code>pos</code>.
 * If <code>pos</code> does not fall in the current buffer, it flushes the
 * current buffer and loads the correct one.<p>
 *
 * On exit from this routine <code>this.curr == this.hi</code> iff <code>pos</code>
 * is at or past the end-of-file, which can only happen if the file was
 * opened in read-only mode.
 */
public void seek(long pos) throws IOException {
    if (pos >= this.hi_ || pos < this.lo_) {
        // seeking outside of current buffer -- flush and read
        this.flushBuffer();
        this.lo_ = pos & BuffMask_; // start at BuffSz boundary
        this.maxHi_ = this.lo_ + (long) this.buff_.length;
        if (this.diskPos_ != this.lo_) {
            super.seek(this.lo_);
            this.diskPos_ = this.lo_;
        }
        int n = this.fillBuffer();
        this.hi_ = this.lo_ + (long) n;
    } else {
        // seeking inside current buffer -- no read required
        if (pos < this.curr_) {
            // if seeking backwards, we must flush to maintain V4
            this.flushBuffer();
        }
    }
    this.curr_ = pos;
}

public long getFilePointer() {
    return this.curr_;
}

public long length() throws IOException {
    // max accounts for the case where we have written past the old file length, but not yet flushed our buffer
    return Math.max(this.curr_, super.length());
}

public int read() throws IOException {
    if (this.curr_ >= this.hi_) {
        // test for EOF
        // if (this.hi < this.maxHi) return -1;
        if (this.hitEOF_)
            return -1;

        // slow path -- read another buffer
        this.seek(this.curr_);
        if (this.curr_ == this.hi_)
            return -1;
    }
    byte res = this.buff_[(int) (this.curr_ - this.lo_)];
    this.curr_++;
    return ((int) res) & 0xFF; // convert byte -> int
}

public int read(byte[] b) throws IOException {
    return this.read(b, 0, b.length);
}

public int read(byte[] b, int off, int len) throws IOException {
    if (this.curr_ >= this.hi_) {
        // test for EOF
        // if (this.hi < this.maxHi) return -1;
        if (this.hitEOF_)
            return -1;

        // slow path -- read another buffer
        this.seek(this.curr_);
        if (this.curr_ == this.hi_)
            return -1;
    }
    len = Math.min(len, (int) (this.hi_ - this.curr_));
    int buffOff = (int) (this.curr_ - this.lo_);
    System.arraycopy(this.buff_, buffOff, b, off, len);
    this.curr_ += len;
    return len;
}

public void write(int b) throws IOException {
    if (this.curr_ >= this.hi_) {
        if (this.hitEOF_ && this.hi_ < this.maxHi_) {
            // at EOF -- bump "hi"
            this.hi_++;
        } else {
            // slow path -- write current buffer; read next one
            this.seek(this.curr_);
            if (this.curr_ == this.hi_) {
                // appending to EOF -- bump "hi"
                this.hi_++;
            }
        }
    }
    this.buff_[(int) (this.curr_ - this.lo_)] = (byte) b;
    this.curr_++;
    this.dirty_ = true;
    syncNeeded_ = true;
}

public void write(byte[] b) throws IOException {
    this.write(b, 0, b.length);
}

public void write(byte[] b, int off, int len) throws IOException {
    while (len > 0) {
        int n = this.writeAtMost(b, off, len);
        off += n;
        len -= n;
        this.dirty_ = true;
        syncNeeded_ = true;
    }
}

/*
 * Write at most "len" bytes to "b" starting at position "off", and return
 * the number of bytes written.
 */
private int writeAtMost(byte[] b, int off, int len) throws IOException {
    if (this.curr_ >= this.hi_) {
        if (this.hitEOF_ && this.hi_ < this.maxHi_) {
            // at EOF -- bump "hi"
            this.hi_ = this.maxHi_;
        } else {
            // slow path -- write current buffer; read next one
            this.seek(this.curr_);
            if (this.curr_ == this.hi_) {
                // appending to EOF -- bump "hi"
                this.hi_ = this.maxHi_;
```

```
            }
        }
    }
    len = Math.min(len, (int) (this.hi_ - this.curr_));
    int buffOff = (int) (this.curr_ - this.lo_);
    System.arraycopy(b, off, this.buff_, buffOff, len);
    this.curr_ += len;
    return len;
  }
}
```

三.测试

1.FileUtil

用于封装三种方案（LineNumberReader、RandomAccessFile、BufferedRandomAccessFile）的文件读取

Java代码

```
package com.gqshao.file.util;

import com.google.common.collect.Lists;
import com.google.common.collect.Maps;
import com.gqshao.file.io.BufferedRandomAccessFile;
import org.apache.commons.io.IOUtils;
import org.apache.commons.lang3.StringUtils;

import java.io.*;
import java.util.List;
import java.util.Map;

public class FileUtil {

    /**
     * 通过BufferedRandomAccessFile读取文件,推荐
     *
     * @param file     源文件
     * @param encoding 文件编码
     * @param pos      偏移量
     * @param num      读取量
     * @return pins文件内容，pos当前偏移量
     */
    public static Map<String, Object> BufferedRandomAccessFileReadLine(File file, String encoding, long pos, int num) {
        Map<String, Object> res = Maps.newHashMap();
        List<String> pins = Lists.newArrayList();
        res.put("pins", pins);
        BufferedRandomAccessFile reader = null;
        try {
            reader = new BufferedRandomAccessFile(file, "r");
            reader.seek(pos);
            for (int i = 0; i < num; i++) {
                String pin = reader.readLine();
                if (StringUtils.isBlank(pin)) {
                    break;
                }
                pins.add(new String(pin.getBytes("8859_1"), encoding));
            }
            res.put("pos", reader.getFilePointer());
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            IOUtils.closeQuietly(reader);
        }
        return res;
    }

    /**
     * 通过RandomAccessFile读取文件，能出来大数据文件，效率低
     *
     * @param file     源文件
     * @param encoding 文件编码
     * @param pos      偏移量
     * @param num      读取量
     * @return pins文件内容，pos当前偏移量
     */
    public static Map<String, Object> readLine(File file, String encoding, long pos, int num) {
        Map<String, Object> res = Maps.newHashMap();
        List<String> pins = Lists.newArrayList();
        res.put("pins", pins);
        RandomAccessFile reader = null;
        try {
            reader = new RandomAccessFile(file, "r");
            reader.seek(pos);
            for (int i = 0; i < num; i++) {
                String pin = reader.readLine();
                if (StringUtils.isBlank(pin)) {
                    break;
                }
                pins.add(new String(pin.getBytes("8859_1"), encoding));
            }
            res.put("pos", reader.getFilePointer());
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            IOUtils.closeQuietly(reader);
        }
        return res;
    }

    /**
     * 使用LineNumberReader读取文件，1000w行比RandomAccessFile效率高，无法处理1亿条数据
     *
     * @param file     源文件
     * @param encoding 文件编码
     * @param index    开始位置
     * @param num      读取量
     * @return pins文件内容
     */
    public static List<String> readLine(File file, String encoding, int index, int num) {
        List<String> pins = Lists.newArrayList();
        LineNumberReader reader = null;
        try {
            reader = new LineNumberReader(new InputStreamReader(new FileInputStream(file), encoding));
            int lines = 0;
            while (true) {
                String pin = reader.readLine();
                if (StringUtils.isBlank(pin)) {
                    break;
                }
                if (lines >= index) {
                    if (StringUtils.isNotBlank(pin)) {
                        pins.add(pin);
                    }
```

```
            }
            if (num == pins.size()) {
                break;
            }
            lines++;
        }
    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        IOUtils.closeQuietly(reader);
    }
    return pins;
}


}
```

2.RandomAccessFileTest

测试方法，涉及到的randomFile只是一个掺杂中文的文本文件，可以自己随便写一个

Java代码

```
package com.gqshao.file;

import com.gqshao.file.util.FileUtil;
import org.apache.commons.collections.CollectionUtils;
import org.apache.commons.collections.MapUtils;
import org.apache.commons.io.IOUtils;
import org.junit.Test;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.*;
import java.util.List;
import java.util.Map;

public class RandomAccessFileTest {

    private static final Logger logger = LoggerFactory.getLogger(RandomAccessFileTest.class);

    private static final String ENCODING = "UTF-8";
    private static final int NUM = 50000;

    private static File file = new File(ClassLoader.getSystemResource("").getPath() + File.separator + "test.txt");
    private static File randomFile = new File(ClassLoader.getSystemResource("").getPath() + File.separator + "RandomFile.txt");

    /**
     * 生成1000w随机文本文件
     */
    @Test
    public void makePin() {
        String prefix = "_$#";
        OutputStreamWriter out = null;
        try {
            out = new OutputStreamWriter(new FileOutputStream(file, true), ENCODING);
            // 在1500w里随机1000w数据
            for (int j = 0; j < 100000000; j++) {
                out.write(prefix + (int) (130000000 * Math.random()) + "\n");
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            IOUtils.closeQuietly(out);
        }
        logger.info(file.getAbsolutePath());
    }

    /**
     * 测试RandomAccessFile读取文件
     */
    @Test
    public void testRandomAccessRead() {
        long start = System.currentTimeMillis();
//
        logger.info(String.valueOf(file.exists()));
        long pos = 0L;
        while (true) {
            Map<String, Object> res = FileUtil.readLine(file, ENCODING, pos, NUM);
            // 如果返回结果为空结束循环
            if (MapUtils.isEmpty(res)) {
                break;
            }
            Object po = res.get("pins");
            List<String> pins = (List<String>) res.get("pins");
            if (CollectionUtils.isNotEmpty(pins)) {
//              logger.info(Arrays.toString(pins.toArray()));
                if (pins.size() < NUM) {
                    break;
                }
            } else {
                break;
            }
            pos = (Long) res.get("pos");
        }
        logger.info(((System.currentTimeMillis() - start) / 1000) + "");
    }

    /**
     * 测试RandomAccessFile读取文件
     */
    @Test
    public void testBufferedRandomAccessRead() {
        long start = System.currentTimeMillis();
//
        logger.info(String.valueOf(file.exists()));
        long pos = 0L;
        while (true) {
            Map<String, Object> res = FileUtil.BufferedRandomAccessFileReadLine(file, ENCODING, pos, NUM);
            // 如果返回结果为空结束循环
            if (MapUtils.isEmpty(res)) {
                break;
            }
            List<String> pins = (List<String>) res.get("pins");
            if (CollectionUtils.isNotEmpty(pins)) {
//              logger.info(Arrays.toString(pins.toArray()));
                if (pins.size() < NUM) {
                    break;
                }
            } else {
                break;
            }
            pos = (Long) res.get("pos");
        }
        logger.info(((System.currentTimeMillis() - start) / 1000) + "");
    }

    /**
     * 测试普通读取文件
     */
    @Test
```

```
    public void testCommonRead() {
        long start = System.currentTimeMillis();
        logger.info(String.valueOf(randomFile.exists()));
        int index = 0;
        while (true) {
            List<String> pins = FileUtil.readLine(file, ENCODING, index, NUM);
            if (CollectionUtils.isNotEmpty(pins)) {
//              logger.info(Arrays.toString(pins.toArray()));
                if (pins.size() < NUM) {
                    break;
                }
            } else {
                break;
            }
            index += NUM;
        }
        logger.info(((System.currentTimeMillis() - start) / 1000) + "");
    }
}
```


- ECMAScript 研究记录 (7)
- CSS 研究记录 (9)
- Maven 研究记录 (8)
- SQL 随笔 (5)
- 权限控制和单点登陆 (8)
- Hadoop 研究记录 (6)
- 随想杂谈 (33)
- JAVA EE (4)
- 测试 (3)
- Redis (10)
- Memcached (2)
- MongoDB (6)
- ElasticSearch (3)

社区版块

- 我的资讯 (0)
- 我的论坛 (77)
- 我的问答 (25)

存档分类

- 2018-05 (3)
- 2018-03 (1)
- 2018-02 (1)
- 更多存档...

最新评论

- sgq0085： 无尘灬 写道楼主，在吗？可以加你qq咨询一下问题吗？公司禁用Q ...
  Shiro通过Redis管理会话实现集群
- 无尘灬： 楼主，在吗？可以加你qq咨询一下问题吗？
  Shiro通过Redis管理会话实现集群
- zhouminsen： 感谢楼主的无私奉献
  Shiro通过Redis管理会话实现集群
- tonny1228： 经测试还是运行在local
  远程调用执行Hadoop Map/Reduce
- asdhobby： 楼主，个人感觉每次调用SessionDAO的doUpdate方 ...
  Shiro通过Redis管理会话实现集群
