链滴

大文件小内存排序问题（阿里笔试题）

作者：valarchie

原文链接：https://ld246.com/article/1593086331387

来源网站：链滴

关于大文件小内存的问题很常见，今天心血来潮想试试自己写一下这个代码。

1.生成4G的数字文本文件

通过计算每行的字节大小，累加到4G的话就停止写入

```
/**
 * 生成一个4G的文件
 *
 * @param file
 * @throws IOException
 */
public static void makeBigFile(File file) throws IOException {

    FileOutputStream fos = new FileOutputStream(file);

    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(fos, "utf-8"));
    // 随机数生成种子
    Random rand = new Random(1);

    long fileByteLength = 0;
    // BIG_FILE_SIZE为4G
    while (fileByteLength < BIG_FILE_SIZE) {

        String randomIntStr = rand.nextInt(1000000000) + "";
        bw.write(randomIntStr);
        bw.newLine();

        // 文件大小等于数字字符串的字符串字节大小加上换行符为两个字节
        fileByteLength += randomIntStr.getBytes().length + 2;

    }

    bw.close();

}
```

2.将大文件分割为256mb的小文件在内存中进行排序

当超过256mb文件时，即更换另一个新的文件。(第一次写，条件没设置好，结果生成了一百多万个文件，删除花了N久)

```
/**
 * 将大文件拆分为不同的小文件
 *
 * @param bigFile 4G大文件
 * @throws IOException
 */
public static void splitBigFileToSmallFile(File bigFile, String dirStr) throws IOException {

    // 提前创建目录
    File dir = new File(dirStr);
```

```
    if (!dir.exists()) {
        dir.mkdirs(); //创建目录
    }

    BufferedInputStream fis = new BufferedInputStream(new FileInputStream(bigFile));

    // 用5M的缓冲读取文本文件
    BufferedReader reader = new BufferedReader(new InputStreamReader(fis, "utf-8"), BUFFER
SIZE);

    String line = null;

    int fileNum = 1;
    long smallFileByteLength = 0;


    File smallFile = new File(dirStr + fileNum + ".txt");
    FileOutputStream fos = new FileOutputStream(smallFile);

    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(fos, "utf-8"));

    while ((line = reader.readLine()) != null) {

        smallFileByteLength += getRowByteLength(line);

        bw.write(line);
        bw.newLine();

        // 当文件超过固定大小的话，进行拆分
        if (smallFileByteLength > FILE_LIMIT_SIZE) {

            bw.flush();

            // 文件数递增
            fileNum++;
            // 小文件的字节大小重置为0
            smallFileByteLength = 0;

            smallFile = new File(dirStr + fileNum + ".txt");
            fos = new FileOutputStream(smallFile);
            bw = new BufferedWriter(new OutputStreamWriter(fos, "utf-8"));

        }

        // 避免代码错误生成非常多小文件
        if (fileNum > 20) {
            break;
        }

    }


    bw.flush();
```

```
    bw.close();

}
```

3.对每个小文件进行内存排序，并重新写入到文件中

不能采用ArrayList，因为当扩容的时候，旧数组和新的扩容数组会导致内存溢出

```
 /**
 * 对每个小文件内的数字进行排序
 *
 * @param file
 * @throws IOException
 */
public static void sortEverySmallFile(File file) throws IOException {

    // ArrayList涉及扩容  我们直接使用数组 否则数组扩容需要拷贝的时候 内存会溢出
    BufferedInputStream countFis = new BufferedInputStream(new FileInputStream(file));

    // 用5M的缓冲读取文本文件
    BufferedReader countReader = new BufferedReader(new InputStreamReader(countFis, "utf
8"), BUFFER_SIZE);

    // 统计行数，便于声明数组的长度
    int rows = 0;

    while (countReader.readLine() != null) {
        rows++;
    }

    // 由于统计完行数的话，reader下标没法重置，所以我们重新载入文件给新的reader
    BufferedInputStream fis = new BufferedInputStream(new FileInputStream(file));
    // 用5M的缓冲读取文本文件
    BufferedReader reader = new BufferedReader(new InputStreamReader(fis, "utf-8"), BUFFER
SIZE);

    int[] numbers = new int[rows];

    for (int i = 0; i < numbers.length; i++) {
        numbers[i] = Integer.valueOf(reader.readLine());
    }

    System.out.println("开始排序");

    Arrays.sort(numbers);
    // 将排序好的数组写入到小文件中
    writeSortToSmallFile(file, numbers);

}
```

4.合并已排序的小文件

我们可以把已排序的小文件看成是数组，借鉴合并两个有序数组的思路，我们这次采用合并16个有序组的思路进行合并。

```
/**
 * 将排序好的小文件进行归并排序
 * 思路是借鉴 两个有序数组的合并
 * 这次是合并16个有序数组
 *
 * @param smallFiles
 * @throws IOException
 */
public static void mergeSmallFilesToBigFile(File[] smallFiles, String sortedBigFilePath) throws
OException {

    // 小文件数组为空的话 直接返回
    if (smallFiles == null || smallFiles.length == 0) {
        return;
    }


    // 为每个小文件生成reader读取每行数字
    BufferedReader[] smallFileReaders = new BufferedReader[smallFiles.length];

    for (int i = 0; i < smallFiles.length; i++) {

        BufferedInputStream fis = new BufferedInputStream(new FileInputStream(smallFiles[i]));
        // 用固定的缓冲读取文本文件
        BufferedReader smallFileReader = new BufferedReader(new InputStreamReader(fis, "utf-
"), BUFFER_SIZE);
        smallFileReaders[i] = smallFileReader;

    }


    // 每一轮比对的最小数字
    Integer[] minArrays = new Integer[smallFiles.length];

    // 预填充
    for (int i = 0; i < smallFileReaders.length; i++) {

        String line = smallFileReaders[i].readLine();

        if (line != null) {
            minArrays[i] = Integer.valueOf(line);
        }

    }

    // 排序好的大文件
    FileOutputStream sortedFos = new FileOutputStream(sortedBigFilePath);
    BufferedWriter sortedBw = new BufferedWriter(new OutputStreamWriter(sortedFos, "utf-8"
, BUFFER_SIZE);
```

```
    long writeNumber = 0;

    // 依次比对每个小文件的最小数字，如果是最小数字的话，继续找下一行
    while (!isAllNull(minArrays)) {

        int minReaderIndex = 0;
        int minNumber = Integer.MAX_VALUE;

        for (int i = 0; i < minArrays.length; i++) {

            if (minArrays[i] != null && minArrays[i] < minNumber) {
                minNumber = minArrays[i];
                minReaderIndex = i;
            }

        }


        sortedBw.write(minNumber + "");
        sortedBw.newLine();

        // 被挑选到最小数字的Reader往下移一行
        String line = smallFileReaders[minReaderIndex].readLine();

        if (line != null) {
            minArrays[minReaderIndex] = Integer.valueOf(line);
        } else {
            minArrays[minReaderIndex] = null;
        }

        writeNumber++;

        if (writeNumber % 10000000 == 0) {
            System.out.println("写入一千万数字");
        }

    }

    // 强制刷出
    sortedBw.flush();

    sortedBw.close();

}
```

5.完整流程

```
 public static void main(String[] args) throws IOException {

        // 先生成一个4G的文件  内包含随机文字
        File file = new File(DESKTOP_DIR + "bigFile.txt");
```

```
        try {
            makeBigFile(file);
        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("生成大文件成功！");

        // 分割大文件变为小文件
        splitBigFileToSmallFile(file, DESKTOP_SMALL_DIR);

        System.out.println("分割大文件成功！");

        // 对生成的小文件进行排序并写入，利用256mb内存进行排序
        File smallFileDir = new File(DESKTOP_SMALL_DIR);

        File[] smallFiles = smallFileDir.listFiles();

        if (smallFiles != null && smallFiles.length > 0) {

            for (File smallFile : smallFiles) {

                sortEverySmallFile(smallFile);
                System.out.println("完成" + smallFile.getName() + "的排序！");

            }

        }

        String sortedBigFilePath = DESKTOP_DIR + "sortedBigFile.txt";

        // 对已排好序的小文件进行归并
        mergeSmallFilesToBigFile(smallFiles, sortedBigFilePath);

        System.out.println("对已排序的小文件合并成功！");

        // 由于文本工具一般没办法打开4g的文件，于是将排序好的文件重新分割成小文件，校验是否
确
        splitBigFileToSmallFile(new File(sortedBigFilePath), DESKTOP_SORTED_SMALL_DIR);

        System.out.println("重新分割已排序的大文件成功！");


    }
```

因为JVM内存区域本身占了一些内存，于是我将内存调至-Xmx290M，如果要严格限制在256mb的，可以将FILE_LIMIT_SIZE调至比256mb更低（将会大于16个文件）。完整代码详见