**课程编号：A0801040060**

**操作系统-大作业 二**



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **姓名** | **柳成林** | **班级学号** | **软件2001**  **20206722** | **成绩** |  |
| **报告名称** | **大作业二** | | | | |
| **开设学期** | **2021-2022春季学期** | | | | |
| **开设时间** | **第16周——第17周** | | | | |
| **使用主要技术** | **主要用到的技术是：**  **在作业1 的基础上使用MappedByteBuffer 内存映射**  注：以上内容，请同学们按照自己作业的实际情况填写 | | | | |
| **可选录制视频** | **☑是 □否**  注：以上内容，请同学们按照自己作业的实际情况填写 | | | | |
| **选做部分** | | | | | |
| **是否做了选做内容** | **☑**（选做）你的系统在转账过程中，如果程序崩溃或系统断电，如何保证持久化数据是正确的（一致的）。  **☑**（选做）你是如何实现步骤2：合并账号余额的。并试图评价你的方法的性能。  **☑**（可选）视频答辩。  **□**没有做可选部分。  注：以上内容，请同学们按照自己作业的实际情况填写 | | | | |

**东北大学软件学院**

如有答辩视频，地址是：

**BiliBili地址：**

[**https://www.bilibili.com/video/BV1uN4y1g716?spm\_id\_from=333.999.0.0**](https://www.bilibili.com/video/BV1uN4y1g716?spm_id_from=333.999.0.0)

**百度网盘地址：**

[**https://pan.baidu.com/s/1h4RoyKkpCJUICz\_WxcicWA?pwd=1234**](https://pan.baidu.com/s/1h4RoyKkpCJUICz_WxcicWA?pwd=1234)

**提取码：1234**

**--来自百度网盘超级会员V1的分享**

**一、实验目的**

**实验内容** **：在大作业part 1银行记账系统基础上完成**

**包括：**

1. 生成数据
2. 合并账号余额
3. 计算转账时间
4. 使用索引文件及内存缓冲等技术，使得转账操作尽可能快
5. 保证程序崩溃时数据的一致性

**二、问题分析与程序设计**

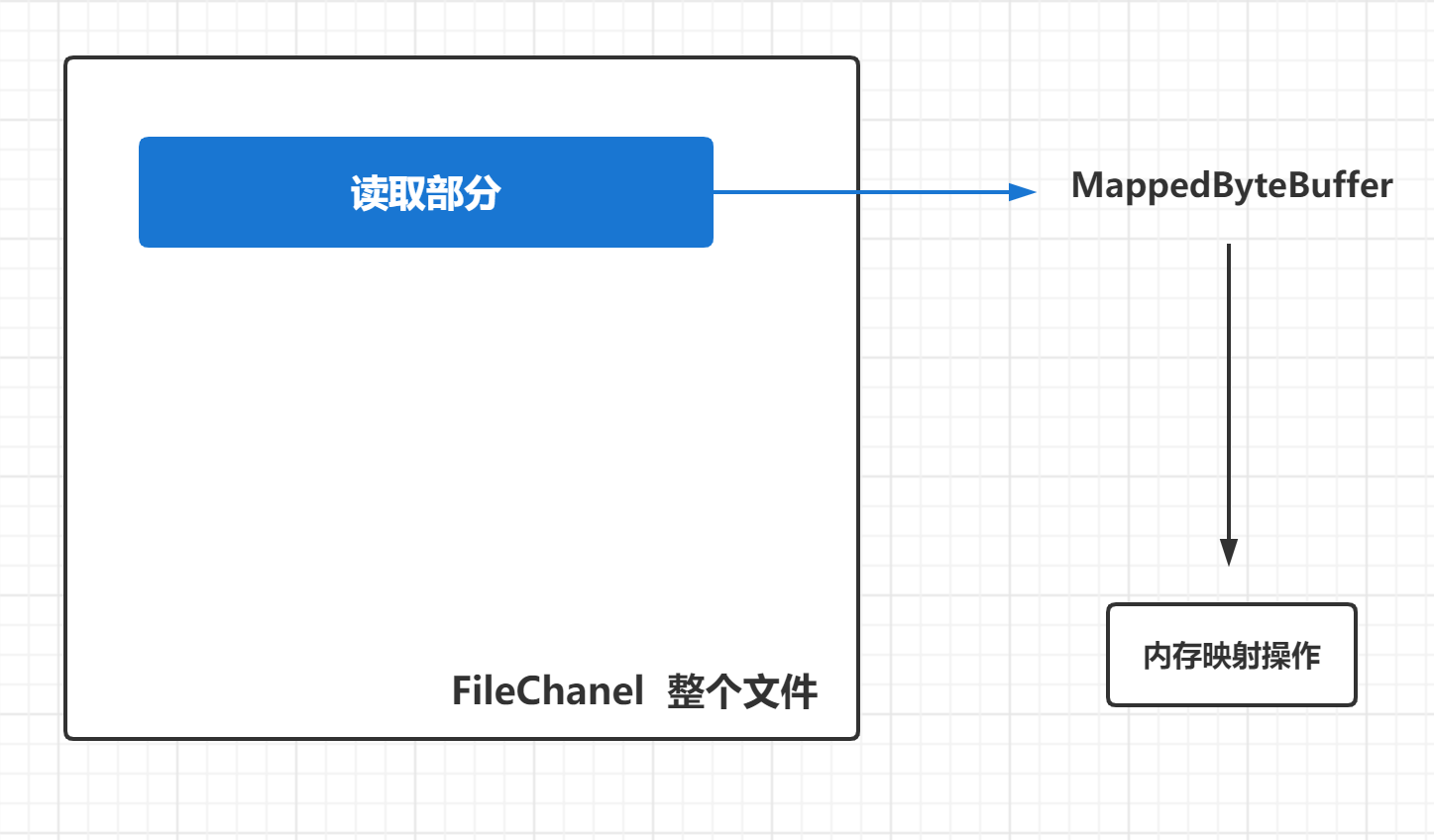
**2.1 大文件存取方法：**

使用MappedByteBuffer。

MappedByteBuffer是ByteBuffer的子类。

以前我们操作大文件都是用BufferedInputStream、BufferedOutputStream等带缓冲的IO流处理。

MappedByteBuffer采用direct buffer的方式读写文件内容,这种方式就是内存映射。这种方式直接调用系统底层的缓存，没有JVM和系统之间的复制操作，所以效率非常高。主要用于操作大文件。

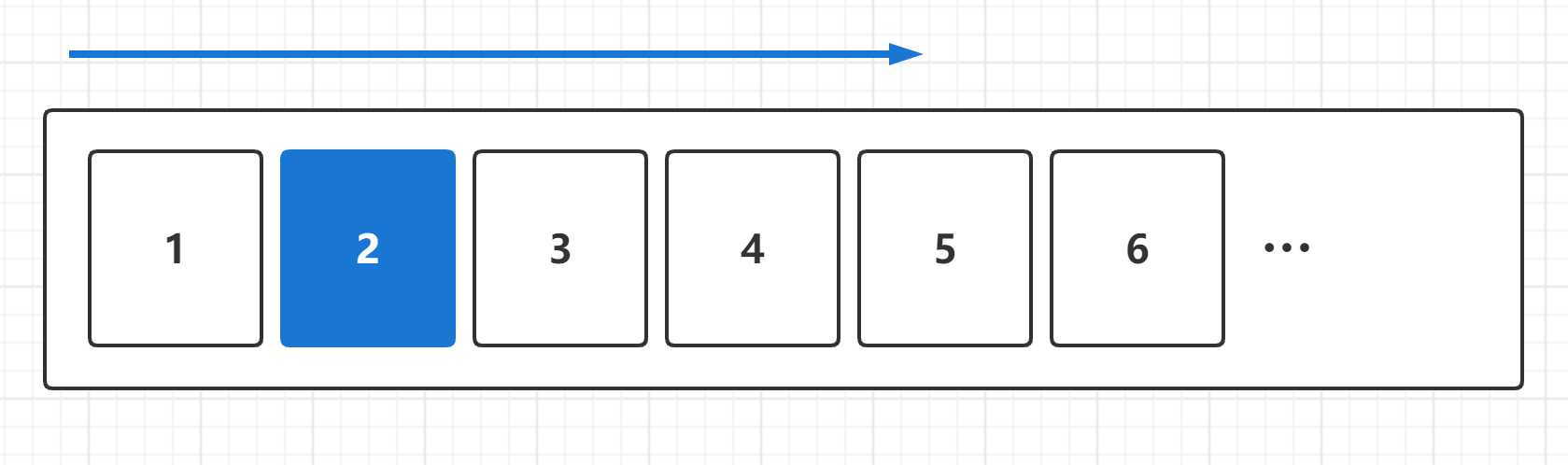


MappedByteBuffer没有构造函数(不可new MappedByteBuffer ()来构造一个MappedByteBuffer()，我们需要借助FileChannel提供的map方法把文件映射为MappedByteBuffer-->MappedByteBuffer map(int mode, long position, long size)；其实就是Map把文件的内容被映像到计算机虚拟内存的一块区域，这样就可以**直接操作内存当中的数据而无需操作的时候每次都通过IO去物理硬盘读取文件，所以效率高。**

具体的读写方式：

**使用了窗口滑动式的内存映射操作读写操作**，窗口滑动式就是建立关联的MappedByteBuffer使用大小为100m，每次只对大文件建立窗口大小的映射。当查询的用户账号很靠后时，需要建立一个循环，检索每一个窗口是否包含该账户，检索完后释放当前窗口，循环关联下一个窗口。（这里速度还能更快，就是同时读取多个窗口，在不同的线程中，当有一个线程找到时就返回true）

MappedByteBuffer采用：MapMode.READ\_WRITE（读/写），对得到的缓冲区的更改最终将写入文件。



**作业1的问题：**

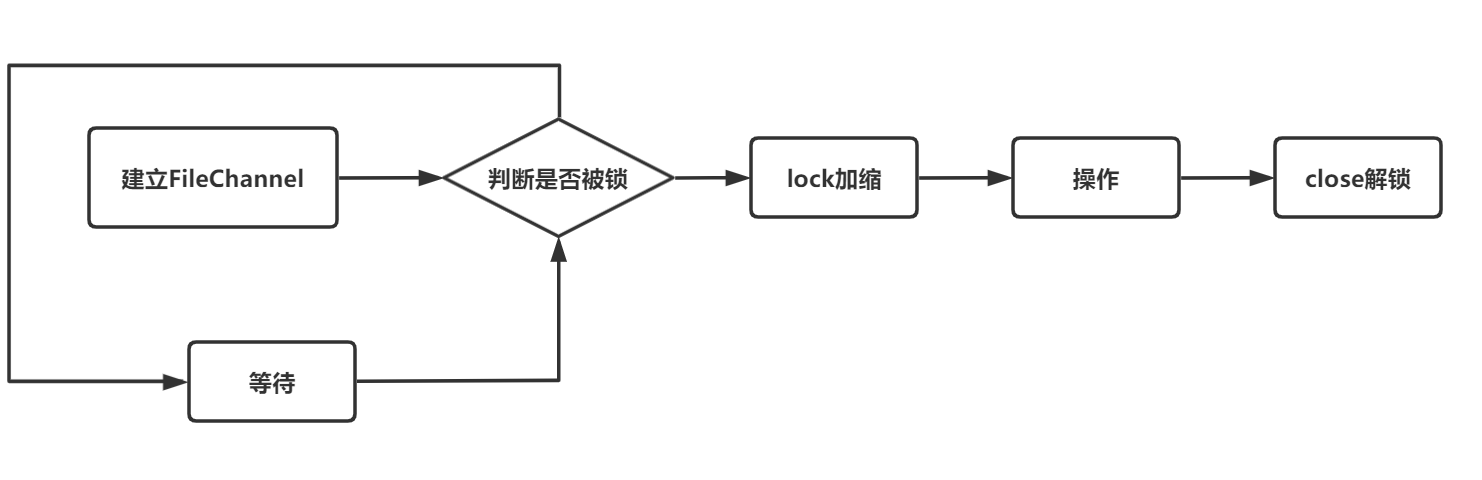
在上次作业中我采用了FileChannel加锁的方式避免文件死锁，这次沿用上次的方法，在文件操作中将数据源替换为account.csv，并把读写文件的方法改为MappedByteBuffer，所以上次作业的问题都可以实现，两次作业并不冲突。

沿用了作业1的方法，但不同点在于方法体使用synchronized，这样可以避免在UserDao对文件进行一直循环判断，与FileChannel结合，当对文件的操作冲突时，会自动延迟执行。



作业1使用的方法，较为繁琐

方法内部逻辑：

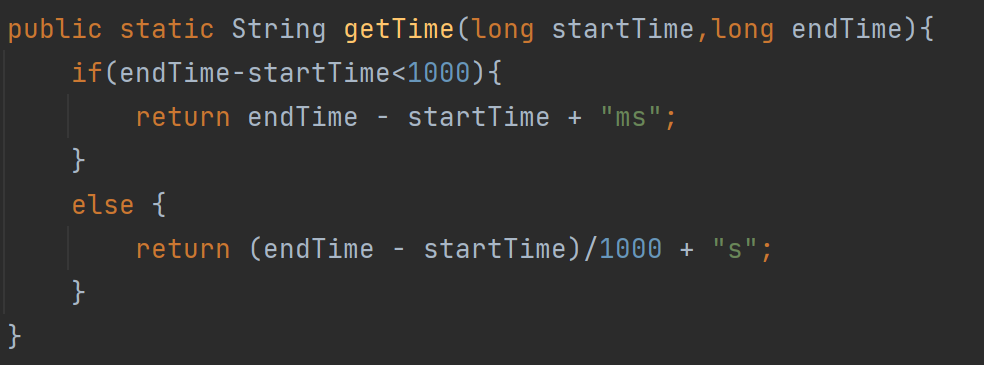


实时问题与批处理问题的解决逻辑没有变化，还是设置原子量。

以上就是对csv进行读写的思路。

**2.2 转账时间记录：**

实现逻辑非常简单，在转账前后获取系统时间即可。



**2.3 崩溃恢复：**

这里我考虑了两种方法：手动恢复与系统优化。

首先考虑崩溃发生会导致系统不一致的原因，由于转账操作是对两个账户进行操作的，必然存在操作顺序问题，假如我们将对一个用户的存取操作定义为原子方法a，那一次转账方法b的执行相当于执行了两次方法a，分别对账户1进行取钱，对账户2进行存钱。

当账户的查找时间较长时，假设平均查找一次账户的时间为5s，那么一次转账的执行时间为10s+，当系统崩溃发生在第6s时，此时账户2没有完成存款操作，只有账户1进行了取钱，这就导致了信息不一致。

手动恢复：

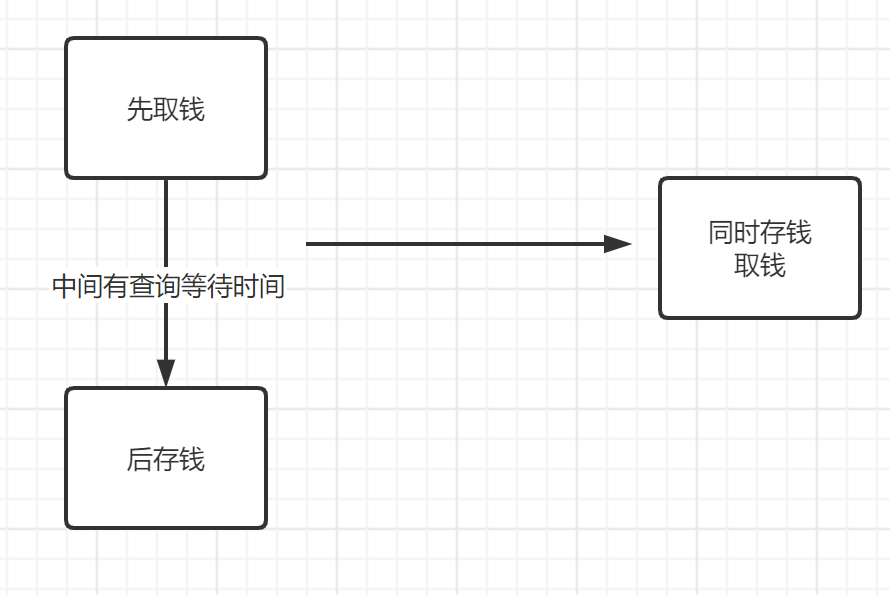
即在每一次操作存款信息时都留有记录。当系统发生崩溃时，用户可以通过该条记录进行数据回滚。

系统优化：

使两次操作同时进行，即先查到两个账户的在csv的位置，然后同时进行存取款。

同时使用MappedByteBuffer也能避免系统崩溃对原文件的损害，因为内存映射形式相当于把本地文件作为内存的一部分处理，文件写入即时生效，就算断电，文件内容数据已经完成持久化了。

这样还能减少总时间，使总时间接近一次存钱。



这里采用第二种方式。

**2.4 合并CSV：**

读取文件方法同上，把账号和金额保存在一个HashMap中，循环判断下一个账号是否已经存在HashMap中，若存在就合并账户。

最后把HashMap的内容写入到新的csv文件里。

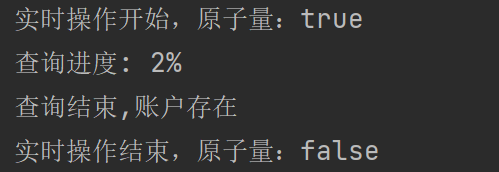
**三、实现结果与测试结果分析**

在上一次作业的基础上加入了记录时间的功能，主界面会显示完成的进程和每一个进程所花费的时间：

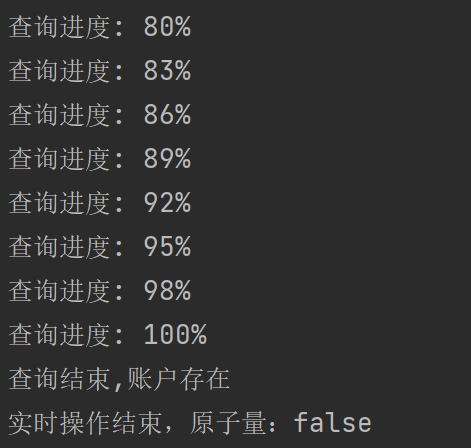


查询进度会随时显示在控制台中，查询进度越小，表名遍历的文件内容越少，速度越快。

查找第一个用户时：



查找最尾用户：



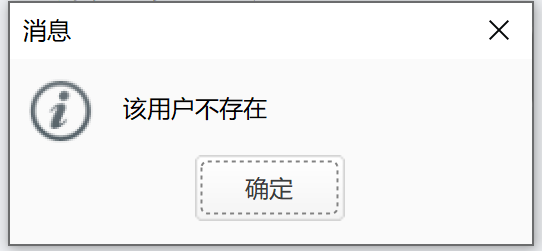
**运行时间：**

****

经过测试，三次转账运行时间分别在15~25s左右。

当查找用户不存在遍历一遍时时间在25s左右。

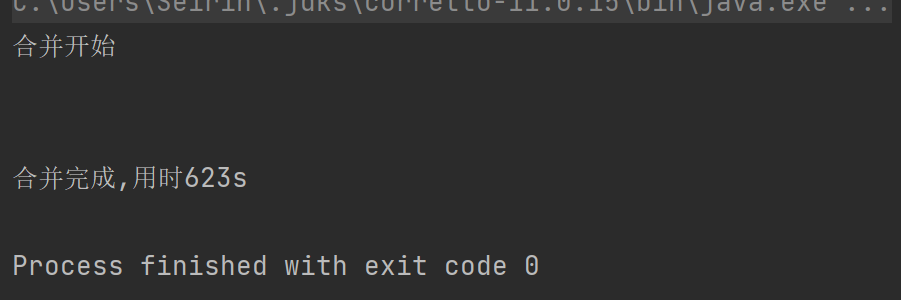
查找不存在时会弹窗提示：



合并CSV：

运行mian2.java

时间大概在10min左右。



关键代码：

 /\*\*  
      \* 实时大文件存取款操作  
      \* @param countName  
      \* @param money  
      \* @return  
      \* @throws IOException  
      \*/  
     public synchronized static double dealMoney(String countName, double money) throws IOException {  
         if (money == 0) {  
             return 0;  
        }  
         FileLock lock = null;  
         FileChannel fileChannel = FileChannel.open(Paths.get(fileName), StandardOpenOption.READ,  
                 StandardOpenOption.WRITE);  
         if (lock == null) {  
             try {  
                 lock = fileChannel.tryLock();  
            }  
             catch (OverlappingFileLockException e){ //如果当前文件被占，报异常  
                 fileChannel.close();  
                 System.out.println("系统异常");  
            }  
        }  
         if (lock != null) {  
             realTime.set(true);  
             System.out.println("实时操作开始，原子量："+realTime.get());  
             long readSize = 0;  
             while (readSize < fileChannel.size()) {  
                 int mSize = BlockSize;  
                 if (readSize + BlockSize > fileChannel.size()) {  
                     mSize = (int) (fileChannel.size() - readSize);  
                }  
                 MappedByteBuffer buffer = fileChannel.map(FileChannel.MapMode.READ\_WRITE, readSize, mSize);  
                 StringBuffer value = new StringBuffer();  
                 while (buffer.hasRemaining()) {  
                     //查找用户,操作金额 此处代码过程略  
                }  
                 buffer.clear();  
                 readSize += mSize;  
                 System.out.println("查询进度: " + 100\*readSize/fileChannel.size() + "%");  
                 if (hasAccount && getMoneySuccess) {  
                     System.out.println("查询结束,账户存在");  
                     break;  
                }  
            }  
             lock.close();  
             realTime.set(false);  
             System.out.println("实时操作结束，原子量："+realTime.get());  
        }  
         fileChannel.close();  
         if (!isAccountExist) {  
             return -2;  
        }  
         return curMoney>=0?curMoney:-1;  
    }  
 ​

合并CSV：

     /\*\*  
      \* 合并CSV  
      \* @param fileName  
      \* @param fileNameOut  
      \*/  
     public void  doCsv(String fileName,String fileNameOut){  
         long startTime = System.currentTimeMillis();  
         //建立Map存放对应的账户和金额  
         Map<String, String> tMap = new HashMap();  
         //使用RandomAccessFile  
         RandomAccessFile file = null;  
         try {  
             file = new RandomAccessFile(new File(fileName), "rw");  
             FileChannel fileChannelIn = file.getChannel();  
             long readSize = 0;  
             //建立  
             FileChannel fileChannelOut = FileChannel.open(Paths.get(fileNameOut), StandardOpenOption.READ,  
                     StandardOpenOption.WRITE, StandardOpenOption.CREATE);  
             //循环遍历整个文件  
             while (readSize < fileChannelIn.size()) {  
                 int mSize = size;  
                 //裁剪  
                 if (readSize + size > fileChannelIn.size()) {  
                     mSize = (int) (fileChannelIn.size() - readSize);  
                }  
                 MappedByteBuffer buffer = fileChannelIn.map(FileChannel.MapMode.READ\_WRITE, readSize, mSize);  
                 //使用StringBuffer,速度更快  
                 StringBuffer key = new StringBuffer();  
                 StringBuffer value = new StringBuffer();  
                 boolean findKey = true;  
                 while (buffer.hasRemaining()) {  
                     byte snyc = buffer.get();  
                     //如果是分隔符,需要进行操作  
                     if (snyc == ',') {  
                         findKey = false;  
                    } else if (snyc == '\n') {  
                         if (key.length() > 0 && value.length() > 0) {  
                             //如果已经有该key 合并  
                             if (tMap.containsKey(key)) {  
                                 double number = Double.parseDouble(tMap.get(key)) + Double.parseDouble(value.toString());  
                                 tMap.put(key.toString(), String.format("%d", number));  
                            } else   
                             //如果没有,新加  
                            {  
                                 tMap.put(key.toString(), value.toString());  
                            }  
                        }  
                         //操作完成后释放空间  
                         key.setLength(0);  
                         value.setLength(0);  
                         findKey = true;  
                    } else {  
                         //不是分隔符,需要连接字符串  
                         if (findKey) {  
                             key.append(Character.toString(snyc)) ;  
                        } else {  
                             value.append(Character.toString(snyc));  
                        }  
                    }  
                }  
                 //写入新文件  
                 toNew(String fileNameOut);  
            }  
             fileChannelOut.close();  
             fileChannelIn.close();  
             long endTime = System.currentTimeMillis();  
             System.out.printf("合并完成,用时%ds", (endTime - startTime)/1000);  
        } catch (Exception e) {  
             e.printStackTrace();  
        }  
    }  
 ​

其他代码详见代码文件。

**四、总结**

这次作业使我巩固了操作系统的知识，同时学习了内存和文件相关的知识。

第一次接触大文件读取，也花费了很多时间在网络上学习。