作业

4．编程题

（1）使用 RandomAccessFile 流将一个文本文件倒置读出。

（2）使用 Java 的输入、输出流将一个文本文件的内容按行读出，每读出一行就顺序添加行号，并写入到另一个文件中。

（3）参考例子 16，解析一个文件中的价格数据，并计算平均价格，该文件的内容如下。

```
商品列表:
电视机,2567 元/台
洗衣机,3562 元/台
冰箱,6573 元/台
```

相关例子：

10.6 随机流

扫一扫

微课视频

通过前面的学习我们知道，如果准备读文件，需要建立指向该文件的输入流；如果准备写文件，需要建立指向该文件的输出流。那么，能否建立一个流，通过该流既能读文件也能写文件呢？这正是本节要介绍的随机流。

RandomAccessFile 类创建的流称作随机流，与前面的输入、输出流不同的是，RandomAccessFile 类既不是 InputStream 类的子类，也不是 OutputStram 类的子类。但是 RandomAccessFile 类创建的流的指向既可以作为流的源，也可以作为流的目的地，换句话说，当准备对一个文件进行读写操作时，创建一个指向该文件的随机流即可，这样既可以从这个流中读取文件中的数据，也可以通过这个流写入数据到文件。

以下是 RandomAccessFile 类的两个构造方法。

- RandomAccessFile（String name,String mode）参数 name 用来确定一个文件名，给出创建的流的源，也是流目的地。参数 mode 取 r（只读）或 rw（可读写），决定创建的

流对文件的访问权利。

- RandomAccessFile（File file,String mode）参数 file 是一个 File 对象，给出创建的流的源，也是流目的地。参数 mode 取 r（只读）或 rw（可读写），决定创建的流对文件的访问权利。

注：RandomAccessFile 流指向文件时，不刷新文件。

RandomAccessFile 类中有一个方法 seek(long *a*)用来定位 RandomAccessFile 流的读写位置，其中参数 *a* 确定读写位置距离文件开头的字节个数。另外流还可以调用 getFilePointer()方法获取流的当前读写位置。RandomAccessFile 流对文件的读写比顺序读写更为灵活。

例子 8 中把几个 int 型整数写入到一个名字为 tom.dat 文件中，然后按相反顺序读出这些数据。

例子 8

Example10_8.java

```
import java.io.**;
public class Example10_8 {
  public static void main(String args[]) {
     RandomAccessFile inAndOut = null;
     int data[] = {1,2,3,4,5,6,7,8,9,10};
     try{  inAndOut = new RandomAccessFile("tom.dat","rw");
         for(int i=0;i<data.length;i++) {
            inAndOut.writeInt(data[i]);
         }
         for(long i=data.length-1;i>=0;i--) {
                                    //一个 int 型数据占 4 个字节，inAndOut 从
            inAndOut.seek(i*4);     //文件的第 36 个字节读取最后面的一个整数
            System.out.printf("\t%d",inAndOut.readInt());
                                    //每隔 4 个字节往前读取一个整数
         }
         inAndOut.close();
     }
     catch(IOException e){}
  }
}
```

表 10.1 是 RandomAccessFile 流的常用方法。

表 10.1　RandomAccessFile 类的常用方法

方法	描述
close()	关闭文件
getFilePointer()	获取当前读写的位置
length()	获取文件的长度
read()	从文件中读取一个字节的数据
readBoolean()	从文件中读取一个布尔值，0 代表 false；其他值代表 true
readByte()	从文件中读取一个字节
readChar()	从文件中读取一个字符（2 个字节）

方法	描述
readDouble()	从文件中读取一个双精度浮点值（8 个字节）
readFloat()	从文件中读取一个单精度浮点值（4 个字节）
readFully(byte b[])	读 b.length 字节放入数组 b，完全填满该数组
readInt()	从文件中读取一个 int 值（4 个字节）
readLine()	从文件中读取一个文本行
readLong()	从文件中读取一个长型值（8 个字节）
readShort()	从文件中读取一个短型值（2 个字节）
readUnsignedByte()	从文件中读取一个无符号字节（1 个字节）
readUnsignedShort()	从文件中读取一个无符号短型值（2 个字节）
readUTF()	从文件中读取一个 UTF 字符串
seek(long position)	定位读写位置
setLength(long newlength)	设置文件的长度
skipBytes(int *n*)	在文件中跳过给定数量的字节
write(byte b[])	写 b.length 个字节到文件
writeBoolean(boolean v)	把一个布尔值作为单字节值写入文件
writeByte(int v)	向文件写入一个字节
writeBytes(String s)	向文件写入一个字符串
writeChar(char c)	向文件写入一个字符
writeChars(String s)	向文件写入一个作为字符数据的字符串
writeDouble(double v)	向文件写入一个双精度浮点值
writeFloat(float v)	向文件写入一个单精度浮点值
writeInt(int v)	向文件写入一个 int 值
writeLong(long v)	向文件写入一个长型 int 值
writeShort(int v)	向文件写入一个短型 int 值
writeUTF(String s)	写入一个 UTF 字符串

需要注意的是，RondomAccessFile 流的 readLine()方法在读取含有非 ASCII 字符的文件时（比如含有汉字的文件）会出现“乱码”现象，因此，需要把 readLine()读取的字符串用“iso-8859-1”编码重新编码存放到 byte 数组中，然后再用当前机器的默认编码将该数组转化为字符串，操作如下。

❶ 读取

```
String str = in.readLine();
```

❷ 用“iso-8859-1”重新编码

```
byte b[] = str.getBytes("iso-8859-1");
```

❸ 使用当前机器的默认编码将字节数组转化为字符串

```
String content = new String(b);
```

如果机器的默认编码是“GB2312”，那么

```
String content = new String(b);
```

等同于

```
String content=new String(b, "GB2312");
```

例子 9 中 RondomAccessFile 流使用 readLine()读取文件。

例子 9

Example10_9.java

```
import java.io.*;
public class Example10_9 {
  public static void main(String args[]) {
    RandomAccessFile in = null;
    try{  in = new RandomAccessFile("Example10_9.java","rw");
        long length = in.length();     //获取文件的长度
        long position = 0;
        in.seek(position);             //将读取位置定位到文件的起始
        while(position<length) {
          String str = in.readLine();
          byte b[] = str.getBytes("iso-8859-1");
          str = new String(b);
          position = in.getFilePointer();
          System.out.println(str);
        }
     }
     catch(IOException e){}
   }
}
```

数字就是单词。下面的例子 16 使用正则表达式（匹配所有非数字字符串）String regex="[^0123456789.]+" 作为分隔标记解析 student.txt 文件中的学生成绩，并计算平均成绩（程序运行效果如图 10.9 所示）。以下是文件 student.txt 的内容。

student.txt

张三的成绩是 72 分,李四成绩是 69 分,刘小林的成绩是 95 分。

```
C:\ch10>java Example10_16
72.0
69.0
95.0
平均成绩:78.66666666666667
```

图 10.9　使用正则表达式解析文件

例子 16

Example10_16.java

```
import java.io.*;
import java.util.*;
public class Example10_16 {
  public static void main(String args[]) {
    File file = new File("student.txt");
    Scanner sc = null;
    int count=0;
    double sum=0;
    try {  double score=0;
         sc = new Scanner(file);
         sc.useDelimiter("[^0123456789.]+");
         while(sc.hasNextDouble()){
            score = sc.nextDouble();
            count++;
            sum = sum+score;
            System.out.println(score);
         }
         double aver = sum/count;
         System.out.println("平均成绩:"+aver);
    }
    catch(Exception exp){
       System.out.println(exp);
    }
  }
}
```