链滴

017 二进制 I/O

作者：pzs233

原文链接：https://ld246.com/article/1537287729353

来源网站：链滴

17.1 引言

文件可分为 文本 或者 二进制 的。

17.2 在 Java 中如何处理文本 I/O

使用 Scanner 类读取文本数据，使用 PrintWrite 类写文本数据。

写入数据：

```
PrintWrite output = new PrintWrite("temp.txt");

output.print("Java 101");

outpout.close();
```

读取数据：

```
Scanner input = new Scanner(new File("temp.txt");
System.out.println(input.nextLine());
```

17.3 文本 I/0 与 二进制 I/O

二进制 I/O 不涉及编码和解码，因此比文本 I/O 更高效。

17.4 二进制 I/O 类

抽象类 InputStream 是读取二进制数据的根类，抽象类 OutputStream 是写入二进制数据的根类。

```
java.io.InputStream 抽象类的方法：
+read(): int
+read(b: byte[]): int
+read(b: btye[], off: int, len: int): int
+available(): int
+close(): void
+skip(n: long): long
+markSupported(): boolean
+mark(readlimit: int): void
+reset(): void
```

```
java.io.OutputStream 抽象类的方法：
+write(int b): void
+write(b: byte[]): void
+write(b: byte[], off: int, len: int): void
+close(): void
+flush(): void
```

FileInputStream 和 FileOutputStream

FileInputStream 类和 FileOutputStream 类用于从/向 文件 读取/写入字节：

java.io.FileInputStream 类（继承 java.io.InputStream）的构造方法:
+FileInputStream(file: File)
+FileInputStream(filename: String)

java.io.FlieOutputStream 类（继承 java.io.OutputStream）的构造方法：
+FileOutputStream(file: file)
+FileOutputStream(filename: String)
+FileOutputStream(file: file, append: boolean)
+FileOutputStream(filename: String, append: boolean)

可以创建一个 PrinterWtiter对象来向文件中追加文本。如果 temp.txt 不存在，就会创建这个文件。果 temp.txt 已经存在，就将新数据追加到该文件中。

new PrintWriter(new FileOutputStream("temp.txt", true));

FilterInputStream 和 FilterOutputStream

FilterInputStream 和 FilterOutputStream：过滤数据的基类。需要处理基本数值类型时，就使用 DataInputStream 和 DataOutputStream 类来过滤字节。

DataInputStream 和 DataOutputStream

<<interface>> java.io.DataInput：
+readBoolean(): boolean
+readByte(): byte
+readChar(): char
+readFloat()： float
+readDouble(): double
+redInt()： int
+readLong(): long
+readShort(): short
+readLine(): String
+readUTF(): String

<<interface>> java.io.DataOutput:
+writeBoolean(b: boolean): void
+writeByte(v: int): void
+writeBytes(s: String): void
+writeChar(c: char): void
+writeChars(s: String): void
+writeFloat(v: float): void
+writeDouble(v: double): void
+writeInt(v: int): void
+writeLong(v: long): void
+writeShort(v: short): void
+writeUTF(s: String): void

创建 DataInputStream 类和 DataOutputStream 类：

public DataInputStream(InputStream instream)
public DataOutputStream(OutputStream outstream)

创建数据流：

```
DataInputStream input = new DataInputStream(new FileInputStrem("in.dat");
DataOutputStream output = new DataOutputStream(new FileOutputStream("out.dat");
```

BufferedInputStream 类和 BufferedOutputStream 类

BufferedInputStream 类和 BufferedOutputStream 类可以通过减少磁盘读写次数来提高输入输出速度。

```
java.io.BufferedInputStream:
+BufferedInputStream(in: InputStream)（默认缓冲大小为 512字节）
+BaufferedInputStream(in: InputStream, bufferSize: int)
```

```
java.io.BufferedOutputStream:
+BufferedOutputStream(out: OutputStream)
+BufferedOutputStream(out: OutputStream, bufferSize；int)
```

使用：

```
DataOutputStream output = new DataOutputStream(new BufferedOutputStream(new FileOuputStrem("temp.dat")));
DataInputStream input = new DataInputStream(new BufferedInputStream(new FileInputStrem9"temp.dat")));
```

对于超过 100MB 的大文件，我们将会看到使用缓冲的 I/O 带来的实质性的性能提升。

```
// 命令行执行f复制程序：java Copy source target（给定源文件 source和目标文件 target）
import java.io.*;

public class Copy {
 /** Main method
    @param args[0] for sourcefile
    @param args[1] for target file
  */
 public static void main(String[] args) throws IOException {
   // Check command-line parameter usage
   if (args.length != 2) {
     System.out.println(
       "Usage: java Copy sourceFile targetfile");
     System.exit(1);
   }

   // Check if source file exists
   File sourceFile = new File(args[0]);
   if (!sourceFile.exists()) {
      System.out.println("Source file " + args[0]
        + " does not exist");
      System.exit(2);
   }

   // Check if target file exists
   File targetFile = new File(args[1]);
   if (targetFile.exists()) {
```

```
      System.out.println("Target file " + args[1]
        + " already exists");
      System.exit(3);
    }

    try (
      // Create an input stream
      BufferedInputStream input =
        new BufferedInputStream(new FileInputStream(sourceFile));

      // Create an output stream
      BufferedOutputStream output =
        new BufferedOutputStream(new FileOutputStream(targetFile));
    ) {
      // Continuously read a byte from input and write it to output
      int r, numberOfBytesCopied = 0;
      while ((r = input.read()) != -1) {
        output.write((byte)r);
        numberOfBytesCopied++;
      }

      // Display the file size
      System.out.println(numberOfBytesCopied + " bytes copied");
    }
  }
}
```

对象 I/O

DataInputStream 类 和 DataOutputStream 类 可以实现基本数据类型与字符串的输入和输出。而 bjectInputStream 类和 ObjectOutputSream 类 除了可以实现基本数据类型与字符串的输入和输出外，还可以用于 读/写 可序列化的对象。

```
java.io.ObjectInputStream:
+ObjectInputStream(in: InputStream)

java.io.ObjectOutputStream:
+ObjectOutputStream(out: OutputStream)

<<interface>> ObjectInput:
+readObject(): Object // 读取一个对象

<<interface>> ObjectOutput:
+writeObject(o: Object): void // 写入一个对象
```

Serializable 接口

并不是每个对象都可以写到输出流。可以写入输出流中的对象称为可序列化的（serializable）。

序列化数组

如果数组中的所有元素都是可序列化的，这个数据就是可序列化的。

一个完整的数组可以用 writeObject 方法存入文件随后用 readObject 方法恢复。

随机访问文件

到现在为止，所使用的所有流都是只读的（read.only）或只写的（write.only）。这些流称为顺序（s quential）流。使用顺序流打开的文件称为 顺序访问文件。顺序访问文件的内容不能更新。

然而，经常需要修改文件。Java 提供了 RandomAccessFile 类，允许在文件的任意位置上进行读写 使用 RandomAccessFile 类打开的文件称为 随机访问文件。

RandomAccessFile 类 实现了 DataInput 和 DataOutput 接口。

Java 提供了 RandomAccessFile 类，允许从文件的任何位置进行数据的读写。

```
java.io.RandomAccessFile:
+RandomAccessFile(file: File, mode: String)
+RandomAccessFile(name: String, mode: String)
+close(): void
+getFilePointer(): long
+length(): long
+read() int
+read(b: byte[], off: int, len: int): int
+seek(pos: long): void
+setLength(newLength: long): void
+skipBytes(int n): int
+write(b: byte[]): void
+write(b: byte[], off: int, len: int): void
```

当创建一个 RandomAccessFile 时，可以指定两种模式（"r"（只读） 或 "rw"（读写））之一。

```
RandomAccessFile raf = new RandomAccessFile("test.dat","rw");
```

如果文件test.dat 已经存在，则创建 raf 以便访问这个文件；如果文件不存在，则创建一个名为 test. at 的新文件，再创建 raf 来访问这个新文件。

设 raf 是 RandomAccessFile 的一个对象，可以调用 raf.seek(position) 方法将文件指针移到指定的 置。 raf.seek(0) 方法将文件指针移到文件的起始位置，而 raf.seek(raf.length) 方法则将文件指针移 文件的末尾。

END

原文链接：017 二进制 I/O