第 3 章

CHAPTER 3

IO 字节输出流

IO 字节输出流通过数据流、序列化、文件系统提供字节输出功能，输入和输出是相对的概念。

3.1 OutputStream 抽象类

此抽象类是所有字节输出流的超类。本节基于官方文档介绍常用方法，核心方法配合代码示例以方便读者理解。

1. close()

关闭当前输出流并释放与此流相关联的任何资源。

2. flush()

清空当前输出流并强制写出任何缓冲的字节。

3. write(byte[] b)

将指定字节数组写出到当前输出流。接收 byte[]入参，作为指定字节数组。

4. write(byte[] b, int off, int len)

将指定字节数组写出到当前输出流。接收 byte[]入参，作为指定字节数组，接收 int 入参，作为数组的起始索引，接收 int 入参，作为写出的最大字节长度。

5. write(int b)

将指定字节数据写出到当前输出流。接收 int 入参，作为指定字节数据。

3.2 FileOutputStream 类

文件字节输出流的作用是将字节数据写出到文件。

3.2.1 构造器

FileOutputStream 构造器见表 3-1。

表 3-1 FileOutputStream 构造器

构 造 器	描 述
FileOutputStream(File file)	构造新的对象,指定文件系统
FileOutputStream(FileDescriptor fdObj)	构造新的对象,指定文件描述符
FileOutputStream(File file, boolean append)	构造新的对象,指定文件系统,指定是否追加内容
FileOutputStream(String name)	构造新的对象,指定文件系统路径
FileOutputStream(String name, boolean append)	构造新的对象,指定文件系统路径,指定是否追加内容

3.2.2 常用方法

本节基于官方文档介绍常用方法,核心方法配合代码示例以方便读者理解。

1. close()

关闭当前输出流并释放与此流相关联的任何资源。

2. flush()

清空当前输出流并强制写出任何缓冲的字节。

3. write(byte[] b)

将指定字节数组写出到当前输出流。接收 byte[]入参,作为指定字节数组,代码如下:

```
//第 3 章/two/FileOutputStreamTest.java
public class FileOutputStreamTest {

    public static void main(String[] args) {
        try (FileOutputStream fileOutputStream = new
                    FileOutputStream("D:\\temp\\src\\test5.txt", false)) {
            String str = "FileOutputStream 文件字节输出流";
            fileOutputStream.write(str.getBytes());
            fileOutputStream.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

运行程序后查看文件内容,如图 3-1 所示。

4. write(byte[] b, int off, int len)

将指定字节数组写出到当前输出流。接收 byte[]入参,作为指定字节数组,接收 int 入参,作为数组的起始索引,接收 int 入参,作为写出的最大字节长度。

5. write(int b)

将指定字节数据写出到当前输出流。接收 int 入参,作为指定字节数据。

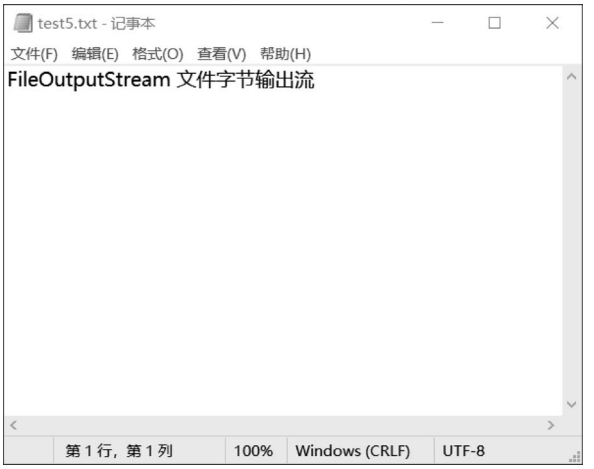

图 3-1 查看文件内容

6. getChannel()

返回与当前文件输出流相关联的唯一文件通道对象。

3.3 ByteArrayOutputStream 类

ByteArrayOutputStream 类包装了一字节数组缓冲区。适用于需要重复使用数据的场景，多线程并发安全。

3.3.1 构造器

ByteArrayOutputStream 构造器见表 3-2。

表 3-2 ByteArrayOutputStream 构造器

构 造 器	描 述
ByteArrayOutputStream()	构造新的对象，默认为无参构造器
ByteArrayOutputStream(int size)	构造新的对象，指定缓冲区的容量大小

3.3.2 常用方法

本节基于官方文档介绍常用方法，核心方法配合代码示例以方便读者理解。

1. close()

关闭当前输出流并释放与此流相关联的任何资源。

2. write(byte[] b)

将指定字节数组写出到当前输出流。接收 byte[]入参，作为指定字节数组，代码如下：

```
//第 3 章/three/ByteArrayOutputStreamTest.java
public class ByteArrayOutputStreamTest {

   public static void main(String[] args) {
      ByteArrayOutputStream byteArrayOutputStream = new
                                                ByteArrayOutputStream();
      System.out.println(byteArrayOutputStream.size());
      byteArrayOutputStream.writeBytes("ByteArrayOutputStream
                  字节缓冲区,并发安全,适用于需要重复使用的数据".getBytes());
      System.out.println(byteArrayOutputStream.size());
   }
}
```

执行结果如下：

```
0
87
```

3. write(byte[] b, int off, int len)

将指定字节数组写出到当前输出流。接收 byte[]入参,作为指定字节数组,接收 int 入参,作为数组的起始索引,接收 int 入参,作为写出的最大字节长度。

4. write(int b)

将指定字节数据写出到当前输出流。接收 int 入参,作为指定字节数据。

5. size()

返回当前缓冲区有效字节的数据长度。

6. reset()

重置当前缓冲区的数据指针,效果类似清空缓冲区的数据。

7. toByteArray()

返回当前缓冲区有效字节数据的副本,代码如下：

```
//第 3 章/three/ByteArrayOutputStreamTest.java
public class ByteArrayOutputStreamTest {
   public static void main(String[] args) {
      ByteArrayOutputStream byteArrayOutputStream = new
                                                ByteArrayOutputStream();
      System.out.println(byteArrayOutputStream.size());
      byteArrayOutputStream.writeBytes("ByteArrayOutputStream
                  字节缓冲区,并发安全,适用于需要重复使用的数据".getBytes());
      System.out.println(byteArrayOutputStream.size());
      try (FileOutputStream fileOutputStream = new
                  FileOutputStream("D:\\temp\\src\\test5.txt", false)) {
         fileOutputStream.write(byteArrayOutputStream.toByteArray());
            //换行符
         fileOutputStream.write(System.lineSeparator().getBytes());
         fileOutputStream.write(byteArrayOutputStream.toByteArray());
```

```
                //换行符
            fileOutputStream.write(System.lineSeparator().getBytes());
            fileOutputStream.write(byteArrayOutputStream.toByteArray());
            fileOutputStream.flush();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

运行程序后查看文件内容，如图 3-2 所示。

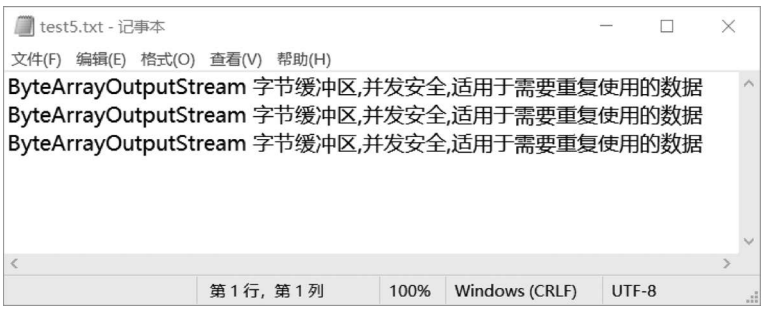

图 3-2　查看文件内容

8. toString()

使用平台的默认字符编码将当前缓冲区内的有效字节数据转换成字符串。

9. toString(Charset charset)

使用指定的字符编码将当前缓冲区内的有效字节数据转换成字符串。接收 Charset 入参，作为指定的字符编码。

10. writeTo(OutputStream out)

将当前缓冲区内的有效字节数据写到指定的输出流。接收 OutputStream 入参，作为指定的输出流。

此方法的源代码如图 3-3 所示。

```
public synchronized void writeTo(OutputStream out) throws IOException {
    out.write(buf, 0, count);
}
```

图 3-3　writeTo(OutputStream out)方法的源代码

3.4　ObjectOutputStream 类

ObjectOutputStream 类需要配合 ObjectInputStream 类使用。提供了序列化、反序列化功能，并提供了字节输出流的相关功能。

3.4.1 构造器

ObjectOutputStream 构造器见表 3-3。

表 3-3 ObjectOutputStream 构造器

构 造 器	描 述
ObjectOutputStream(OutputStream out)	构造新的对象,指定字节输出流

3.4.2 常用方法

本节基于官方文档介绍常用方法,核心方法配合代码示例以方便读者理解。

1. close()

关闭当前输出流并释放与此流相关联的任何资源。

2. write(byte[] buf)

将指定字节数组写出到当前输出流。接收 byte[]入参,作为指定字节数组。

3. write(byte[] b, int off, int len)

将指定字节数组写出到当前输出流。接收 byte[]入参,作为指定字节数组,接收 int 入参,作为数组的起始索引,接收 int 入参,作为写出的最大字节长度。

4. write(int b)

将指定字节数据写出到当前输出流。接收 int 入参,作为指定字节数据。

5. writeBoolean(boolean val)

将指定布尔数据写出到当前输出流。接收 boolean 入参,作为指定布尔数据。

6. writeChar(int val)

将指定字符数据写出到当前输出流。接收 int 入参,作为指定字符数据。

7. writeDouble(double val)

将指定双精度数据写出到当前输出流。接收 double 入参,作为指定双精度数据。

8. writeFloat(float val)

将指定单精度数据写出到当前输出流。接收 float 入参,作为指定单精度数据。

9. writeInt(int val)

将指定整数型数据写出到当前输出流。接收 int 入参,作为指定整数型数据。

10. writeLong(long val)

将指定长整数型数据写出到当前输出流。接收 long 入参,作为指定长整数型数据。

11. writeObject(Object obj)

将指定对象型数据写出到当前输出流。接收 Object 入参，作为指定对象型数据。

12. writeUTF(String str)

将指定字符串数据写出到当前输出流。接收 String 入参，作为指定字符串数据。

注意：此方法入参的字符串数据默认使用 UTF-8 编码进行字节转换。

3.5 字符编码转换工具类

编写字符文件编码转换工具类，支持任意长度的文件数据，代码如下：

```
//第 3 章/fice/FileEncodeUtil.java
public class FileEncodeUtil {

   public static void encodeChange(File srcFile, String srcEncode,
                                    File targetFile, String targetEncode) {
      if (srcFile == null || srcEncode == null
                                    || targetFile == null || targetEncode == null
                                    || !srcFile.exists() || !targetFile.exists()
                                    || !srcFile.isFile() || !targetFile.isFile()
                                    || !Charset.isSupported(srcEncode)
                                    || !Charset.isSupported(targetEncode)
                                    || srcEncode.equals(targetEncode)) {
         System.out.println("必要入参不支持...");
         return;
      }
      //字节数据缓冲区
      ByteBuffer buffer = ByteBuffer.allocate(8192);
      //源文件解码器
      CharsetDecoder charsetDecoder =
                                    Charset.forName(srcEncode).newDecoder();
      //字符数据缓冲区
      CharBuffer charBuffer = CharBuffer.allocate(8192);
      //解码器的解码结果
      CoderResult coderResult;
      try (FileInputStream fileInputStream = new FileInputStream(srcFile);
         FileOutputStream fileOutputStream = new
                                      FileOutputStream(targetFile, false)) {
         //文件是否还有数据可读
         while (fileInputStream.available() > 0) {
            //文件输入字节流 -> 字节数据缓冲区
            int read = fileInputStream.read(buffer.array(),
                                    buffer.position(), buffer.remaining());
            //设置字节数据缓冲区数据的指针位置
```

```
                buffer.position(read + buffer.position());
                //将字节数据缓冲区切换为输出模式
                buffer.flip();
                do {
                    //字节数据缓冲区 -> 解码 -> 字符数据缓冲区
                    coderResult = charsetDecoder.decode(buffer,
                                                    charBuffer, false);
                    if (coderResult.isError()) {
                        throw new UnsupportedEncodingException(
                                                        "解码错误!");
                    }
                    //将字符数据缓冲区切换为输出模式
                    charBuffer.flip();
                    //字符数据缓冲区 -> 字符串 -> 编码成字节 -> 目标文件
                    fileOutputStream.write(charBuffer.toString().
                                getBytes(Charset.forName(targetEncode)));
                    //清空字符数据缓冲区
                    charBuffer.clear();
                    //判断上次解码时字符数据缓冲区是否容量不够
                //此种情况下字节数据缓冲区可能还有数据可以转换
                } while (coderResult.isOverflow());
                //压缩字节数据缓冲区数据,并切换为输入模式
                buffer.compact();
            }
            if (buffer.position() > 0) {
                throw new UnsupportedEncodingException("解码错误!");
            }
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}
```

注意：以上代码用到了 NIO 相关的知识点，后续章节会详细介绍。

小结

字节输出流可以输出任意格式的文件数据，文件名称后缀是一个规范的设计体系。

习题

1. 判断题

(1) 文件字节输出流可以输出任意类型的文件数据。(　　)

（2）文件字节输出流使用完毕后必须关闭以释放资源。（　　）

（3）字符数据代表文字型内容。（　　）

2. 选择题

（1）OutputStream 类可以输出数据的方法有（　　）。（多选）

A. close()　　B. flush()

C. write(byte[] b)　　D. write(int b)

（2）FileOutputStream 类可以将数据强制刷新到文件的方法是（　　）。（单选）

A. close()　　B. flush()

C. write(byte[] b)　　D. write(int b)

（3）可以达到字节缓存效果的类是（　　）。（单选）

A. FileOutputStream　　B. ObjectOutputStream

C. ByteArrayOutputStream　　D. PipedOutputStream

（4）可以序列化对象的类是（　　）。（单选）

A. FileOutputStream　　B. ObjectOutputStream

C. ByteArrayOutputStream　　D. PipedOutputStream

3. 填空题

查看执行结果并补充代码，代码如下：

```
//第 3 章/answer/SerializableTest.java
public class SerializableTest {
   public static void main(String[] args) {
      objectWriter();
      objectReader();
   }

   public static void objectReader() {
      try (FileInputStream fis = new
                              FileInputStream("D:\\temp\\src\\t.txt");
           ObjectInputStream ois = new ObjectInputStream(fis)) {
         int i = ois.readInt();
         String today = (String) ois._____;
         Date date = (Date) ois._____;
         System.out.println(i);
         System.out.println(today);
         System.out.println(date);
      } catch (Exception e) {
         e.printStackTrace();
      }
   }

   public static void objectWriter() {
      try (FileOutputStream fos = new
                              FileOutputStream("D:\\temp\\src\\t.txt");
```

```
            ObjectOutputStream oos = new ObjectOutputStream(fos)) {
            oos.writeInt(12345);
            oos._____________
            oos.writeObject(new Date(454545454999L));
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}
```

执行结果如下：

```
12345
Today
Mon May 28 06:37:34 CST 1984
```