Java 应用程序设计
高级 I/O 编程

王晓东
wxd2870@163.com

中国海洋大学

November 17, 2017

参考书目

1. 张利国、刘伟 [编著], Java SE 应用程序设计, 北京理工大学出版社, 2007.10.

本章学习目标

1. Java I/O 原理
2. 基本 I/O 流类型
3. I/O 应用
4. 对象序列化
5. NI/O

大纲

接下来…

Java I/O 原理

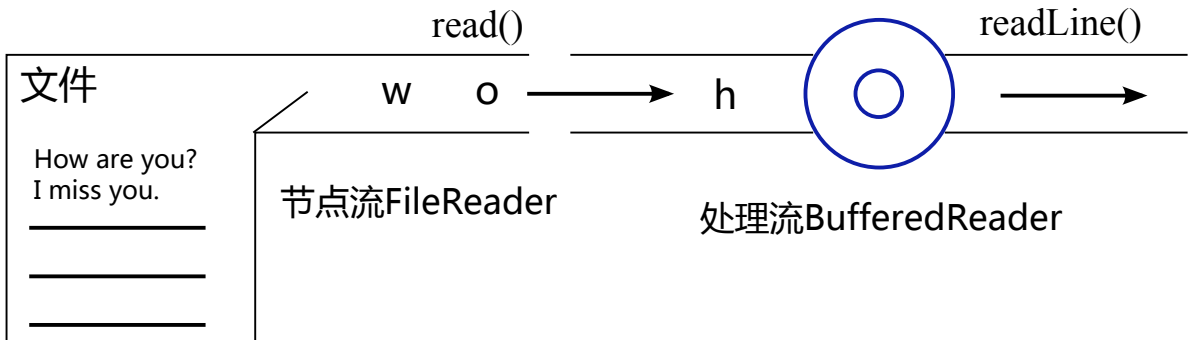
❖ 基本概念

- I/O（Input/Output）
- 数据源（Data Source）
- 数据宿（Data Sink）
- 流（Stream）：Java 中把不同的数据源与程序间的数据传输都抽象表述为流，java.io 包中定义了多种 I/O 流类型实现数据 I/O 功能。

Java I/O 原理

❖ I/O 流的分类

- **按照数据流动的方向，Java 流可分为输入流（Input Stream）和输出流（Output Stream）。**输入流只能从中读取数据，而不能向其写出数据；输出流则只能向其写出数据，而不能从中读取数据。（特例：java.io.RandomAccessFile 类）
- **根据数据流所关联的是数据源还是其他数据流，可分为节点流（Node Stream）和处理流（Processing Stream）。**节点流直接连接到数据源；处理流是对一个已存在的流的连接和封装，通过所封装的流的功能调用实现增强的数据读/写功能，处理流并不直接连到数据源。



Java I/O 原理

❖ I/O 流的分类

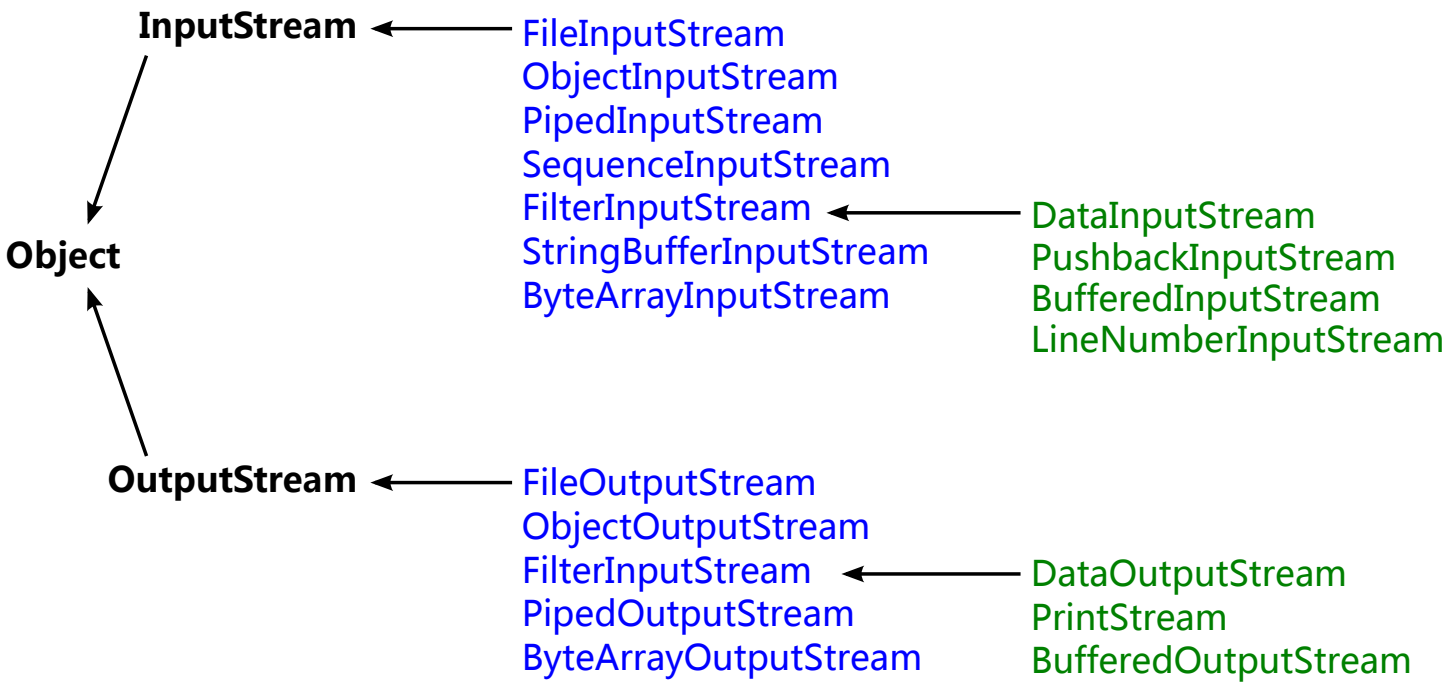
- **按传输数据的“颗粒大小”划分，可分为字符流**（Character Stream）**和字节流**（Byte Stream）。字节流以字节为单位进行数据传输，每次传送一个或多个字节；字符流以字符为单位进行数据传输，每次传送一个或多个字符。
- 从 JDK1.4 版本开始，Sun 公司引入了新的 Java I/O API（NIO, New Input/Output），**提供面向数据块、异步 I/O 操作。**

❖ Java 命名惯例

凡是以 InputStream 或 OutputStream 结尾的类型均为**字节流**，凡是以 Reader 或 Writer 结尾的均为**字符流**。

接下来…

InputStream and OutputStream



InputStream

抽象类 java.io.InputStream 是所有字节输入流类型的父类，该类中定义了以字节为单位读取数据的基本方法，并在其子类中进行了分化和实现。

三个基本的 read 方法

- int read()
- int read(byte[] buffer)
- int read(byte[] buffer, int offset, int length)

其它方法

- void close()
- int available()
- skip(long n)
- boolean markSupported()

OutputStream

java.io.OutputStream 与 java.io.InputStream 对应，是所有字节输出流类型的抽象父类。

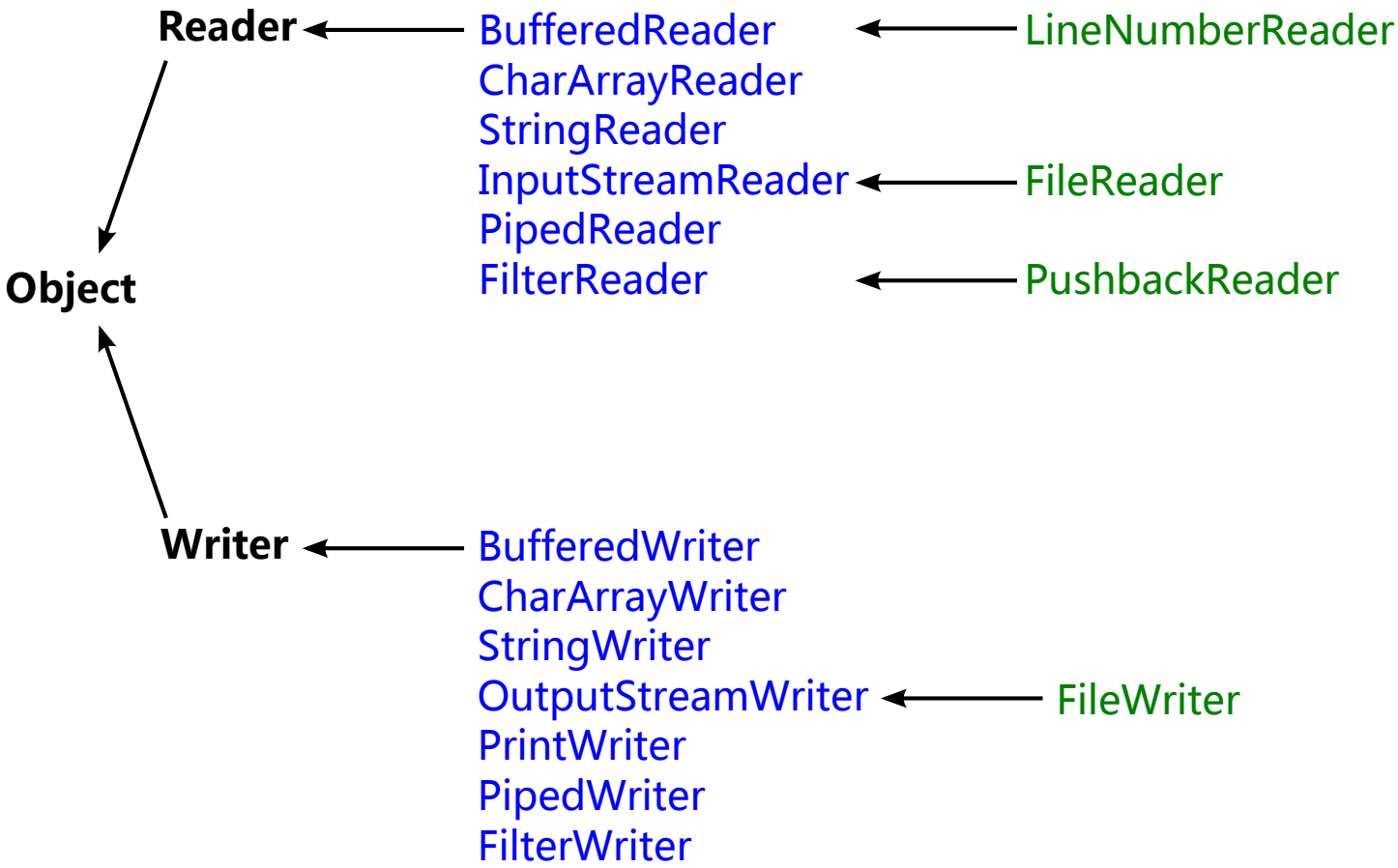
三个基本的 write 方法

- void write(int c)
- void write(byte[] buffer)
- void write(byte[] buffer, int offset, int length)

其它方法

- void close()
- void flush()

Reader and Writer



Reader

抽象类 java.io.Reader 是所有字符输入流类型的父类，其中声明了用于读取字符流的有关方法。

三个基本的 read 方法

- int read()
- int read(char[] cbuf)
- int read(char[] cbuf, int offset, int length)

其它方法

- void close()
- boolean ready()
- skip(long n)
- boolean markSupported()
- void mark(int readAheadLimit)

Writer

java.io.Writer 与 java.io.Reader 类对应，是所有字符输出流类型的共同父类。

五个基本的 write 方法

- void write(int c)
- void write(char[] cbuf)
- void write(char[] cbuf, int offset, int length)
- void write(String string)
- void write(String string, int offset, int length)

其它方法

- void close()

接下来…

FileInputStream/FileOutputStream

- FileInputStream 用于读取本地文件中字节数据，FileOutputStream 用于将字节数据写出到文件。
- FilenputStream 不适合获取文本文件中的字符信息，要读取并显示的文件中如果含有双字节字符（如中文），则会显示乱码，此时应该采用字符流类型。
- *可以用于复制任何格式的文件，如文本、音视频以及可执行文件等二进制文件，因为以字节为单位进行数据复制时并不对文件内容进行解析。*

CODE ◗ Fragment: 使用字节流实现文件复制

```
1 FileInputStream fis = new FileInputStream("in.txt");
2 FileOutputStream fos = new FileOutputStream("out.txt");
3 int read = fis.read();
4 while (read != -1) {
5   fos.write(read);
6   read = fis.read();
7 }
8 fis.close();
9 fos.close();
```

FileReader/FileWriter

- FileReader 用于以**字符**为单位读取文本文件，FileWriter 类用于将字符数据写出到文本文件。
- 字符 I/O 流类型只能处理文本文件，因为二进制文件中保存的字节信息不能正常解析为字符。

CODE ◗ Fragment: **使用字符流实现文件复制**

```
1  FileReader fis = new FileReader("in.txt");
2  // The second arg is boolean append, true for appending, false for covering.
3  FileWriter fos = new FileWriter("out.txt", true);
4  int read = fis.read();
5  while (read != -1) {
6    fos.write(read);
7    read = fis.read();
8  }
9  fis.close();
10 fos.close();
```

BufferedReader/BufferedWriter

- 处理流。
- BufferedReader 用于缓冲读取字符、字符数组或行，采用缓冲处理能够提高效率，该类所封装的字节输入流对象需要在构造方法中指定。
  - public BufferedReader(Reader in)
  - public BufferedReader(Reader in, int size) // size of buffer
- BufferedWriter 提供字符的缓冲写出功能，该类的 newLine() 方法可以写出平台相关的行分隔符来标记一行的终止，此分割符由系统属性 line.separator 确定。

CODE ◗ Fragment: 使用字符处理流实现文件复制

```
1 BufferedReader br = new BufferedReader(new FileReader("in.txt"));
2 BufferedWriter bw = new BufferedWriter(new FileWriter("out.txt"));
3 String s = br.readLine();
4 while (s != null) {
5   bw.write(s);
6   bw.newLine(); // notice.
7   s = br.readLine();
8 }
```

Other I/O Classes

- InputStreamReader/OutputStreamWriter
- PrintStream/PrintWriter
- DataInputStream/DataOutputStream
- CharArrayReader/CharArrayWriter

接下来…

Redirection of Standard I/O

Java 控制台程序默认是以控制台键盘和显示器作为标准输入/输出设备，在有些有情况下，我们可能希望将程序的标准输入或标准输出进行重新定向。

例如，程序测试时可能需要大量数据，如果使用控制台输入测试数据的话每次都要重新输入，这样很繁琐，此时可以考虑进行输入重定向。

Redirection of Standard I/O

CODE ◗ TestSetInput.java

```
1  import java.io.*;
2  public class TestSetInput {
3    public static void main(String[] args) {
4      try {
5        FileInputStream fis = new FileInputStream("in.txt");
6        System.setIn(fis);
7        int avg = 0;
8        int total = 0;
9        int num = 0;
10       int i;
11       BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
12       String s = br.readLine();
13       while (s != null && !s.equals("over")) {
14         i = Integer.parseInt(s);
15         num++;
16         total += i;
17         avg = total / num;
18         System.out.println("num=" + num + "\ttotal=" + total + "\tavg=" + avg);
19         s = br.readLine();
20       }
21     } catch (IOException e) {
22       e.printStackTrace();
23     }
24   }
25 }
```

其中，in.txt 为每行均为整数的文件。

Redirection of Standard I/O

- System.setOut(OutputStream output)
- System.setErr(OutputStream output)

属性信息的导入/导出

如果要永久记录用户自定义的属性，可以采用 Properties 类的 load()/store() 方法进行属性的导入/导出操作，即将属性信息写出到文件中和从文件中读取属性信息到程序。

CODE ◗ SavaProperties.java

```
1  import java.io.FileWriter;
2  import java.util.Properties;
3  public class SaveProperties {
4    public static void main(String[] args) {
5      try {
6        Properties ps = new Properties();
7        ps.setProperty("name", "Kevin");
8        ps.setProperty("password", "12345");
9        FileWriter fw = new FileWriter("props.txt");
10       ps.store(fw, "loginfo");
11       fw.close();
12     } catch (Exception e) {
13       e.printStackTrace();
14     }
15   }
16 }
```

属性信息的导入/导出

CODE ◗ LoadProperties.java

```
1  import java.io.FileWriter;
2  import java.util.Properties;
3  public class LoadProperties {
4    public static void main(String[] args) {
5      try {
6        Properties ps = new Properties();
7        FileReader fr = new FileReader("props.txt");
8        ps.load(fr);
9        fr.close();
10       ps.list(System.out);
11     } catch (Exception e) {
12       e.printStackTrace();
13     }
14   }
15 }
```

File: props.txt

```
#loginfo
#Sun Nov 04 21:20:17 CST 2012
password=12345
name=Kevin
```

Stdout

```
-- listing properties --
password=12345
name=Kevin
```

随机存取文件

❖ 需求

- 存档游戏，记录新玩家的第一次成绩时，添加一条新纪录。
- 老用户重玩时，更新其原有记录信息（不添加新纪录）。
- 浏览所有玩家的记录信息。

❖ 一种实现方式

使用 java.io.RandomAccessFile 类实现文件的随机存取操作，该方法能够同时提供读/写文件的功能。[1]

[1]代码见 tex 源文件

接下来…

Java I/O 原理

基础 I/O 流

常用 I/O 流类型

I/O 应用

对象序列化

对象序列化

❖ 概念

对象序列化（Object Serialization）是指将对象的状态数据以字节流的形式进行处理，一般用于实现对象的持久性，即长久保存一个对象的状态并在需要时获取该对象的信息以重新构造一个状态完全相同的对象。对象序列化可以理解为使用 I/O “对象流”类型实现对象读/写操作。

❖ 基本概念

- **对象的持久性**（Object Persistance） 长久保存一个对象的状态并在需要时获取该对象的信息以重新构造一个状态完全相同的对象。
- **对象序列化**（Object Serialization） 通过写出对象的状态数据来记录一个对象。
- **对象序列化的主要任务** 写出对象的状态信息，并遍历该对象对其他对象的引用，递归的序列化所有被引用到的其他对象，从而建立一个完整的序列化流。

实现对象序列化

要序列化一个对象，其所属的类必须实现以下两种接口之一：

- java.io.Serializable
- java.io.Externalizable

java.io.ObjectOutputStream 和 ObjectInputStream 类分别提供了对象的序列化和反序列化功能。

☞ 注意

- 在对象序列化过程中，其所属类的 static 属性和方法代码不会被序列化；
- 对于个别不希望被序列化的非 static 属性，也可以在属性声明的时候使用 transient 关键字进行标明。

实现对象序列化

CODE ◗ Employee.java

```
1   package test.objectserialization;

3   import java.io.Serializable;

5   public class Employee implements Serializable {

7     private static final long serialVersionUID = 1L;
8     private String name;
9     private int age;
10    private String dept;

12    public Employee() {

14    }

16    public Employee(String name, int age, String dept) {
17      this.name = name;
18      this.age = age;
19      this.dept = dept;
20    }

22    public void showInfo() {
23      System.out.println("Name:␣" + name + "\tAge:␣" + age + "\tDept:␣" + dept);
24    }

26    //other setter and getter methods.

28  }
```

实现对象序列化

CODE ▸ WriteObject.java

```
1   package test.objectserialization;
2
3   import java.io.FileNotFoundException;
4   import java.io.FileOutputStream;
5   import java.io.IOException;
6   import java.io.ObjectOutputStream;
7
8   public class WriteObject {
9
10    public static void main(String[] args) {
11
12      try {
13        FileOutputStream fos = new FileOutputStream("dada.ser");
14        ObjectOutputStream oos = new ObjectOutputStream(fos);
15
16        oos.writeObject(new Employee("Kevin", 35, "OUC␣CS"));
17        oos.writeObject(new Employee("Wang␣Xiaodong", 35, "OUC␣CS"));
18
19        oos.close();
20      } catch (FileNotFoundException e) {
21        e.printStackTrace();
22      } catch (IOException e){
23        e.printStackTrace();
24      }
25    }
26  }
```

实现对象序列化

CODE ▸ ReadObject.java

```
1   package test.objectserialization;
2
3   import java.io.FileInputStream;
4   import java.io.IOException;
5   import java.io.ObjectInputStream;
6
7   public class ReadObject {
8
9     public static void main(String[] args) {
10
11      try {
12        FileInputStream fis = new FileInputStream("dada.ser");
13        ObjectInputStream ois = new ObjectInputStream(fis);
14
15        Employee e1 = (Employee) ois.readObject();
16        Employee e2 = (Employee) ois.readObject();
17
18        e1.showInfo();
19        e2.showInfo();
20
21        ois.close();
22      } catch (ClassNotFoundException e) {
23        e.printStackTrace();
24      } catch (IOException e) {
25        e.printStackTrace();
26      }
27    }
28  }
```

THE END

wxd2870@163.com