第9章 文件与输入输出流

- 9.1 File类与文件操作
- 9.2 输入输出流
- 9.3 字节流
- 9.4 字符流
- 9.5 对象序列化
- 9.6 随机存取文件

9.1 File类与文件操作

输入/输出对应的英文单词是input/output，因此，输入/输出操作通常简称为I/O操作。Java的I/O类和接口主要包含在java.io包中(从JDK 1.4起引入了一些与缓冲区、通道有关的新I/O类库，它们位于java.nio包中)。Java.io包中提供了通过数据流、序列化和文件系统实现输入、输出的功能，如果程序中需要导入其中的类、接口，需要写上import java.io.Xxx;或import java.io.*;语句(Xxx是类或接口名)。

由于受多种因素的影响(如：访问的文件不存在)，I/O操作有可能不成功，通常，需要用try…catch…结构来捕获IOException异常，这一点务必注意。

9.1 File类与文件操作

9.1.1 File类

计算机的操作系统是用路径名来标识文件和目录的，如果我们在编写管理文件程序时也采用这种方式，操作起来并不方便，且路径名依赖于操作系统。为此，Java专门提供了一个类——File来实现这一目标。

File类

◆java.io中的一个类

◆Object的直接子类，其功能是以抽象方式表示文件和目录

◆提供生成文件、目录；修改、删除等方法

9.1 File类与文件操作

9.1.1 File类

构造方法，有三种格式

(1) **File(String pathname)**：参数是文件或目录的路径名，数据类型为String。

(2)**File(String parent, String child)**：第一个参数是父目录，第二个参数为子路径名，两者均为String类型。

(3)**File(File parent, String child)**：与(2)类似，只是第一个参数为File类型。

File类的对象通常用作文件管理、输入输出流类的参数，上述三种格式选用哪一种都可以，关键是要正确标识文件与目录。

9.1 File类与文件操作

9.1.1 File类

类常量

File.Separator：路径分隔符

◆在window中对应“\”

Java解释器把String对象中的反斜杠(\)作为转义符，所以，为了在String对象中引入反斜杠，不得不使用“\\”来代替。

◆在Linux中对应”/”

可以用File.Separator来获取系统的路径分隔符

比如要根据用户输入的文件夹信息来动态地营造文件
String i1= “d:\\temp\\”
String path=i1+”java.txt”

// 假设输入的是 i1=“d:\\temp”呢
String path=i1+”\\”+”java.txt”

If(!i1.EndWith(File.Separator)) i1+=File.Separator

9.1 File类与文件操作

9.1.1 File类

常用方法

File类的方法有几十个，没有必要死记硬背，只要掌握文件/目录操作的几个常用方法，了解主要属性的获取、测试、设置功能即可，其它的使用时查阅API文档。为方便大家理解，我们将这些方法分为几种类型：

(1)获取文件/目录某一属性的值

(2)测试文件/目录是否具备某一属性

(3)设置文件/目录某一属性

(4)文件/目录操作

public boolean exists()	目录文件是否存在
public boolean canRead()	目录文件是否可读
public boolean canWrite()	目录文件是否可写
public String getAbsolutePath()	得到文件的绝对路径
public String getName()	得到文件名
public String getParent()	得到父目录的名字
public String getPath()	返回路径
public nativeboolean isAbsolute()	如果是绝对路径返回真
public boolean isDirectory()	如果是目录则返回真
public boolean isFile()	如果是文件则返回真
public long lastModified()	返回最近一次修改时间
public long length()	返回文件长度

示例

- 编程完成：
  - 编写一个函数实现如下功能，显示指定目录的详细信息：
    - 是否绝对路径
    - 文件的长度
    - 是否可读
    - 是否可写
    - 是否目录
    - 是否文件
    - 是否存在
  - 再编写程序修改其只读属性

9.1 File类与文件操作

9.1.2文件操作

在使用计算机时，我们经常会进行文件/目录的遍历工作

public String[] list(FilenameFilter filter)	**返回指定格式的目录中的文件名**
public String[] list()	**返回当前目录中的所有文件名**
public File[] listFiles(FilenameFilter filter)	**返回某一目录下符合过虑条件的目录和文件列表**
public File[ ] listFiles()	**返回某一目录下所有目录和文件列表**

例子：遍历整个目录

遍历目录

```
void bianli(File dir){
    if (dir.isDirectory()){
        File[] subdir=dir.listFiles();
        for(File f:subdir){
          bianli(f);
        }
    }
    else
       System.out.println(dir.getAbsolutePath());
}
```

Java中的文件

- File类：
- 文件过虑器FilenameFilter类：
  - java.io.FilenameFilter接口中包含一个方法：
    - public boolean accept(File dir,String name)
    - 说明：File类中的list和listFiles方法，取出当前File对象所代表的根目录下的所有子目录和文件，依次调用FilenameFilter对象的accept方法，将代表根目录的File对象和子目录或文件名分别传给accept方法的参数。只有当accept返回true，才会将这个目录或文件加入返回清单中

```
class FilterExample implements FilenameFilter{
    String filtername="";
    FilterExample(String filtername ){
            this.filtername=filtername;
    }
    public boolean accept(File dir, String name){
            if (name.contains(filtername))
                    return true;
            else
                    return false;
            }
    }
    }
```

9.1 File类与文件操作

9.1.2文件操作

在使用计算机时，我们经常会进行文件/目录操作，例如：新建文件/目录；文件/目录改名；删除文件/目录等等。File类提供了丰富的方法，可以实现类似功能。

- 对文件、目录进行的操作类的方法：

public boolean delete()	**删除文件或目录，其中目录为空时才能删**
public URL toUrl()	**将路径名转成URL格式**
public boolean mkdir()	**创建目录，成功返回真**
public boolean mkdirs()	**创建路径中所有目录，成功则返回真**
public boolean renameTo(Filedest)	**文件更名，成功返回真**

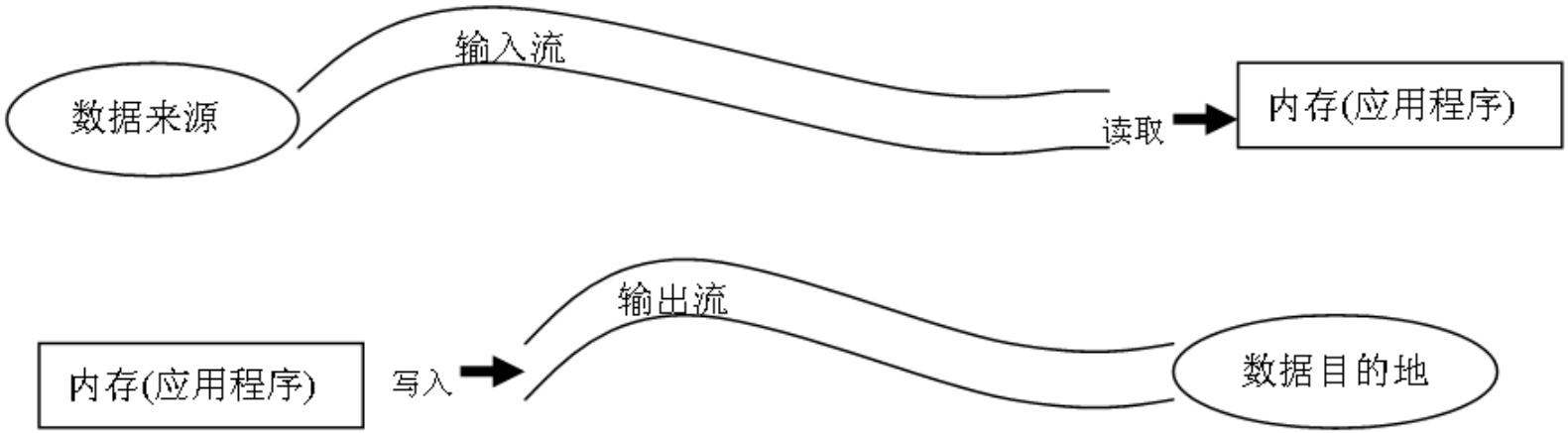
Public boolean createNewFile() **按照拟定的文件名创建一个空文件**

TestFile.java

9.2 输入输出流

9.2.2 输入输出流

“流”的一个重要特征是具有方向性，输入流(input stream)表示数据从输入设备(如键盘、磁盘、网络)流向内存，输出流(output stream)则是数据从内存流向输出设备(如屏幕、磁盘、网络)。**需要牢记的是：应始终站在内存(即应用程序)的角度来区分输入输出流。**



9.2 输入输出流

9.2.2 输入输出流

由于应用程序是从输入流中读取数据(不能向其写入数据)、向输出流写入数据(不能从中读取数据)，所以，输入流和输出流的操作方法有很大不同：

输入流只能进行读取操作

主要掌握read()方法(包括其变形，如readXxx())的使用；

输出流进行的是写入操作

重点关注write()方法(包括其变形，如writeXxx())的使用。

read()、wrtie()等方法大都有几种重载格式，应注意比较它们的差异。

9.2.3 Java中的流及其分类

根据数据处理基本单位的不同，Java中的流又可分为字节流和字符流两种类型。

字节流：是以字节(byte，8位)为基本单位，将数据看作是由一个个字节构成的序列，可处理任何类型的数据(包括二进制数据和文本信息)，这是较低层次的操作。

字符流：是以字符(Unicode编码，16位，2字节)为基本单位，将数据看作是由一个个字符组成的序列，适用于字符、文本类型数据的操作，例如：文本文件的读写，网络聊天信息的传送。输入输出时存在Unicode编码和本地字符集码的转换问题，需要进行编码、解码处理。这是JDK 1.1后为了方便字符处理而增加的内容。

9.2 输入输出流

9.2.3 Java中的流及其分类

对字节流、字符流进行区分是必要和有益的，这是操作输入输出流的基础。

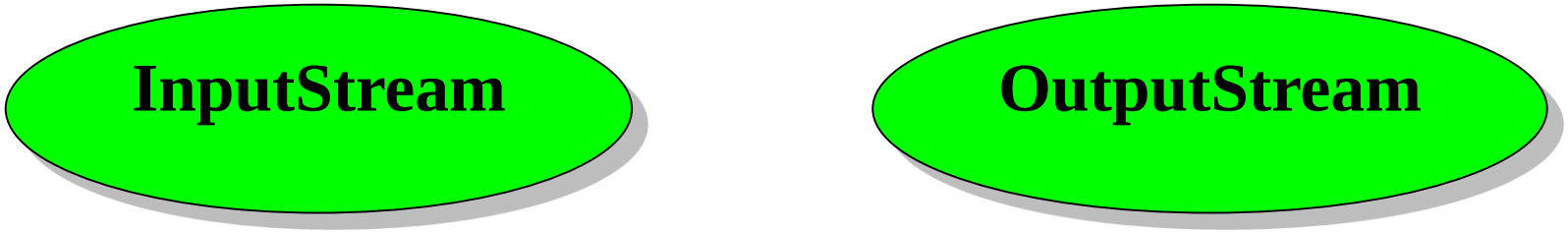
至此，已介绍了按两种不同标准划分的两组流：输入流与输出流、字节流与字符流。

如果将它们交叉起来，就能形成4种流：**字节输入流、字节输出流、字符输入流、字符输出流**，它们各自包含多个子类，能够实现丰富功能。

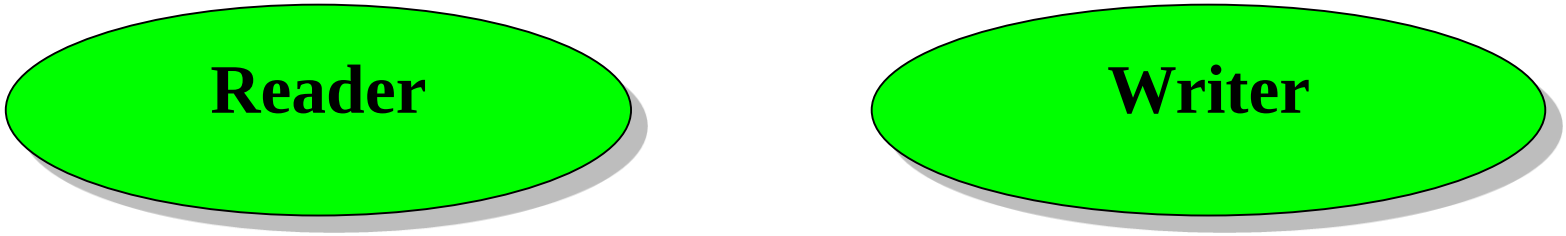
9.2 输入输出流

9.2.3 Java中的流及其分类

1．字节流：字节输入流、字节输出流的基类分别是InputStream、OutputStream。

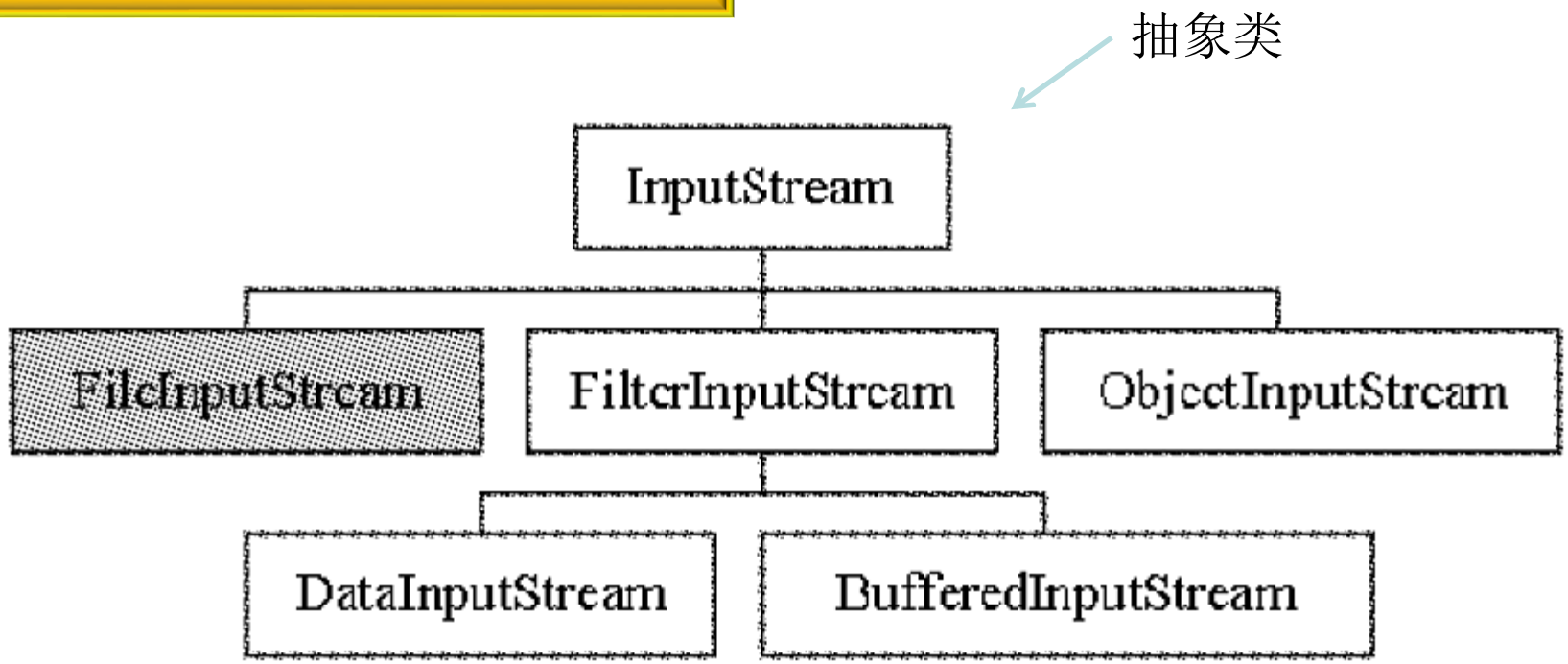


2．字符流：字符输入流、字符输出流的基类分别是Reader、Wirter。



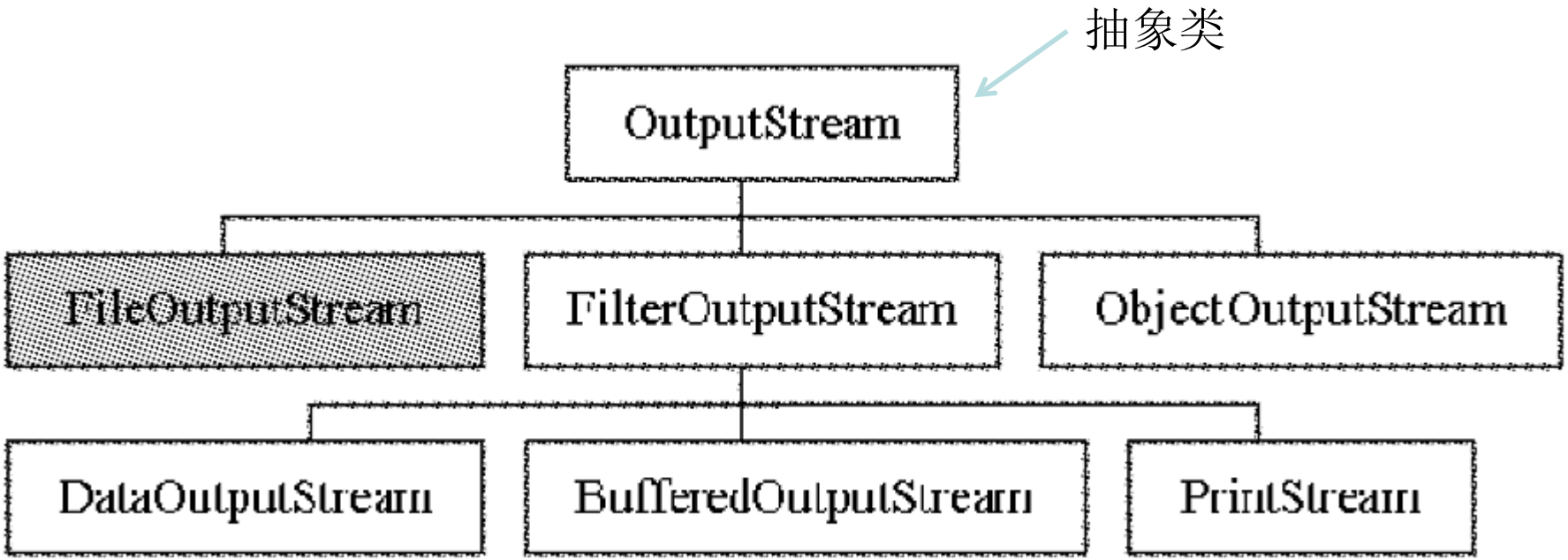
9.3 字节流

9.2.3 Java中的流及其分类



9.3 字节流

9.2.3 Java中的流及其分类



标准输入/输出流

概述

- 程序读写文件时，在读写完毕后，就会及时关闭输入流或输出流——这些输入流、输出流对象的生命周期是短暂的，不会存在程序运行的整个生命周期
- 程序运行的整个生命周期中，可能需要从同一个数据源读入数据，或向同一个目标输出数据——例如日志，用于跟踪用户、程序运行的状态

标准I/O

- 在java.lang.System类中提供了三个静态变量：
  - System.in
    - InputStream类型，**标准输入流**，默认数据源是键盘
    - 程序中可利用System.in读取标准输入流中数据
  - System.out
    - PrintStream类型，**标准输出流**，默认数据汇是控制台
    - 程序可利用System.out输出运行时的正常消息
  - System.err
    - PrintStream类型，**标准错误输出流**，默认数据汇是控制台
    - 程序可利用System.err输出运行时的错误信息

标准I/O

- 标准I/O的说明：
  - System.in、System.out、System.err三种流是由JVM创建的
  - 三种流存在于程序运行的整个生命周期中
  - 这三个流始终处于打开状态，除非程序中显式地关闭了它们

9.2 输入输出流

9.2.3 Java中的流操作一般步骤

在Java中要操作输入输出流，通常按以下步骤来进行：

(1) 引入java.io包中的类；

(2) 打开输入流或输出流；

(3) 从输入流中读取数据或向输出流写入数据；

(4) 关闭流。

9.3.2 文件字节流

1.FileInputStream类构造方法

常用的有两种格式：

(1) FileInputStream(File f)：以File类型为参数构造对象。

(2) FileInputStream(String name)：以String类型为参数构造对象。

代码段1：File f=new File(" d:\\mydir\\readme.txt");

FileInputStream infile1=new FileInputStream(f);

代码段2：FileInputStream infile2=new FileInputStream("d:\\mydir\\readme.txt");

不过，要保证FileInputStream对象所对应的文件存在且可读，否则，会抛出**FileNotFoundException**异常。

9.3.2 文件字节流

1.FileInputStream常用方法

int read()

顺序读取文件中一个字节的内容，如果读到文件尾部，返回-1

int read(byte[] b)

顺序读取b.length个字节的内容存储到数组b中，返回值是实际读取的字节个数 ，如果读到文件尾部，返回-1

看一个读文件的例子（单字节读取）：

```
void readfile1(String filepath) throws IOException{
    FileInputStream fs=null;
    try{
        fs=new FileInputStream(filepath);
        int i=fs.read();//read one byte
        while(i!=-1){
            System.out.print((char)i); //打印出字节对应的字符
            i=fs.read();
        }
    }
    catch(Exception ex){
        ex.printStackTrace();
    }
    finally{
        fs.close();
    }
}
```

See readfile1-FileInputOutput.java

看一个读文件的例子（字节数组方式读）：

```
void readfile2(String filepath) throws IOException{
    FileInputStream fs=null;
    try{
        fs=new FileInputStream(filepath);
        byte[] b=new byte[128];
        int i=fs.read(b);
        while(i!=-1){
            String str=new String(b,0,i);//这样不会产生乱码
            System.out.print(str);
            i=fs.read(b);
        }
    }
    catch(Exception ex){
        ex.printStackTrace();
    }
    finally{
        fs.close();
    }
}
```

See readfile2-FileInputOutput.java

9.3.2 文件字节流

2.FileOutputStream类构造方法

(1) FileOutputStream(File file)：以File类型为参数构造对象，如file指定的文件不存在，将新建一个新文件；若指定的文件存在，将用新内容覆盖原先内容。

(2) FileOutputStream(String name)：以String类型为参数构造对象，其功能与(1)类似。

(3) FileOutputStream(File file, boolean append)：基本功能与(1)类似，只是当append值为true时，新增内容以追加方式放在原内容之后；当append为false时，将用新内容覆盖原先内容。

(4) FileOutputStream(String name, boolean append)：只是参数类型为String，其余与(3)相同。

9.3 字节流

9.3.2 文件字节流

2.FileOutputStream常用方法

void write(byte[] b)
将字节数组b写入到文件中

void write(byte[] b, int off, int len)
将字节数组b从下标off处开始，取长度len个元素写入文件

9.3 字节流

一个小应用：
CopyFile

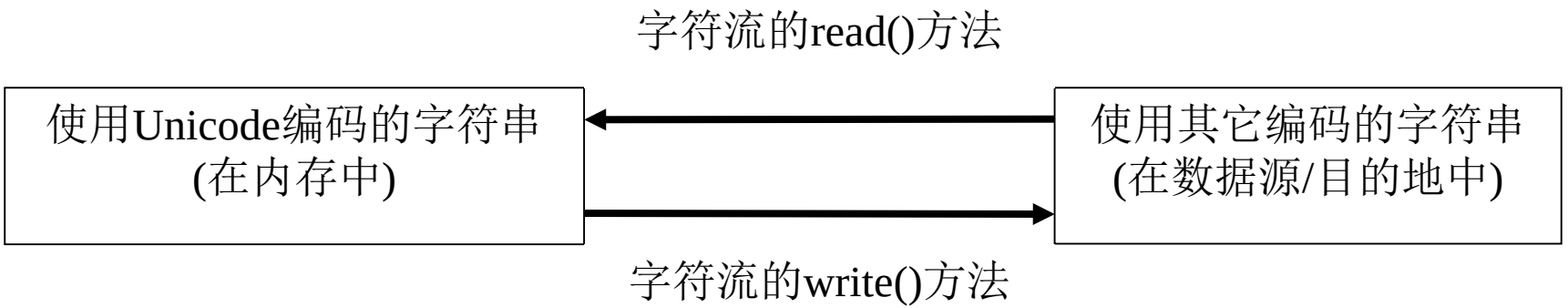
思路：
打开一个指定位置的文件A的全部内容，
在指定位置新建一个同名文件B，
把A的内容读出来写入到B中

See
copyFile-FileInputOutput.java

```
void copyfile(String filepath1,String filepath2){
    FileInputStream fsin=null;
    FileOutputStream fsout=null;
    try{
        fsin=new FileInputStream(filepath1);
        fsout=new FileOutputStream(filepath2);
        byte[] b=new byte[128];
        int i=fsin.read(b);//read  bytes
        while(i!=-1){
            fsout.write(b,0,i);
            i=fsin.read(b);
        }
        fsin.close();
        fsout.close();
    }
    catch(Exception ex){
    ex.printStackTrace();
    }
}
```

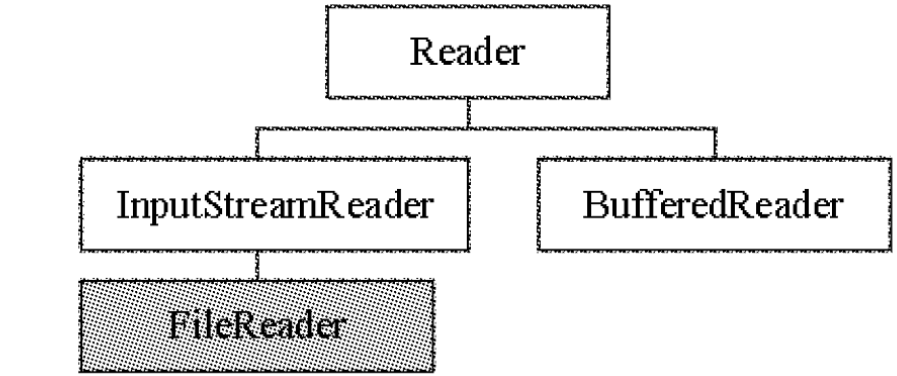
9.4 字符流

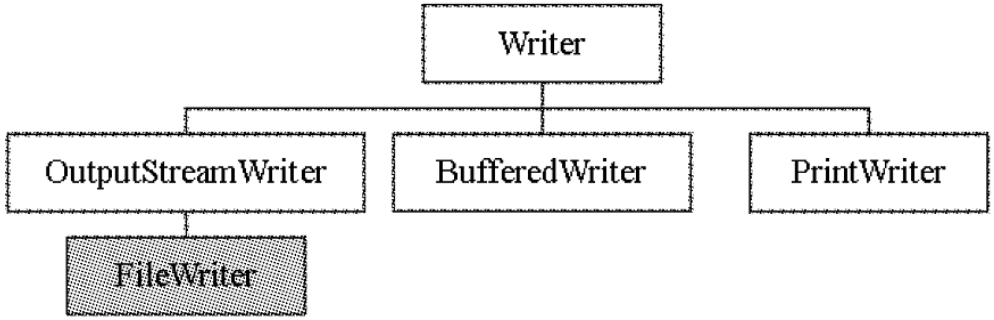
字符串中可能包含汉字，这里有必要说明一下汉字的编码及转换。在数据源/目的地中可能是以Unicode之外的其它字符集编码的，那么，在内存中使用Unicode编码的字符串与数据源/目的地中使用其它编码的字符串是如何转换的呢？这一工作由字符流来完成。



9.4 字符流

字符流中的类也有十几个，我们采用与字节流类似的处理方式，只挑选几个比较实用、有代表性的类进行介绍，这些类的继承关系如图所示。





9.4 字符流

9.4.1 字符流的基类

1.Reader类的基本方法

与字节流类似，字符输入流最重要的功能是“读取”数据，只是操作的基本单位变成了“字符”而已。基本方法如下：

(1)read()：从输入流中读取数据。有3种格式：

(2) void close()：关闭输入流，并释放与该输入流有关的系统资源。

(3) boolean ready()：输入流是否做好读取准备。注意：字符流中无int available()方法。

(4) long skip(long n)：从输入流中跳过n个字符。

(5) void reset()：使输入流读指针重新复位到刚刚标记的位置处。

9.4 字符流

9.4.1 字符流的基类

2.Writer类的基本方法

同样道理，字符输出流的重要功能也是“写入”数据，操作单位改为“字符”。基本方法如下：

(1)write()：向输出流写入数据。

(2) void close()：关闭输出流，并释放与该输出流相关的系统资源。

(3) void flush()：将缓冲区中的数据强制进行写操作，刷新输出缓冲区。

9.4 字符流

9.4.2 InputStreamReader和OutputStreamWriter类

1.常用的构造方法

(1) InputStreamReader(InputStream in)：使用系统默认的字符集生成字符输入流。

(2) InputStreamReader(InputStream in, String charsetName)：使用用户指定的字符集生成字符输入流。

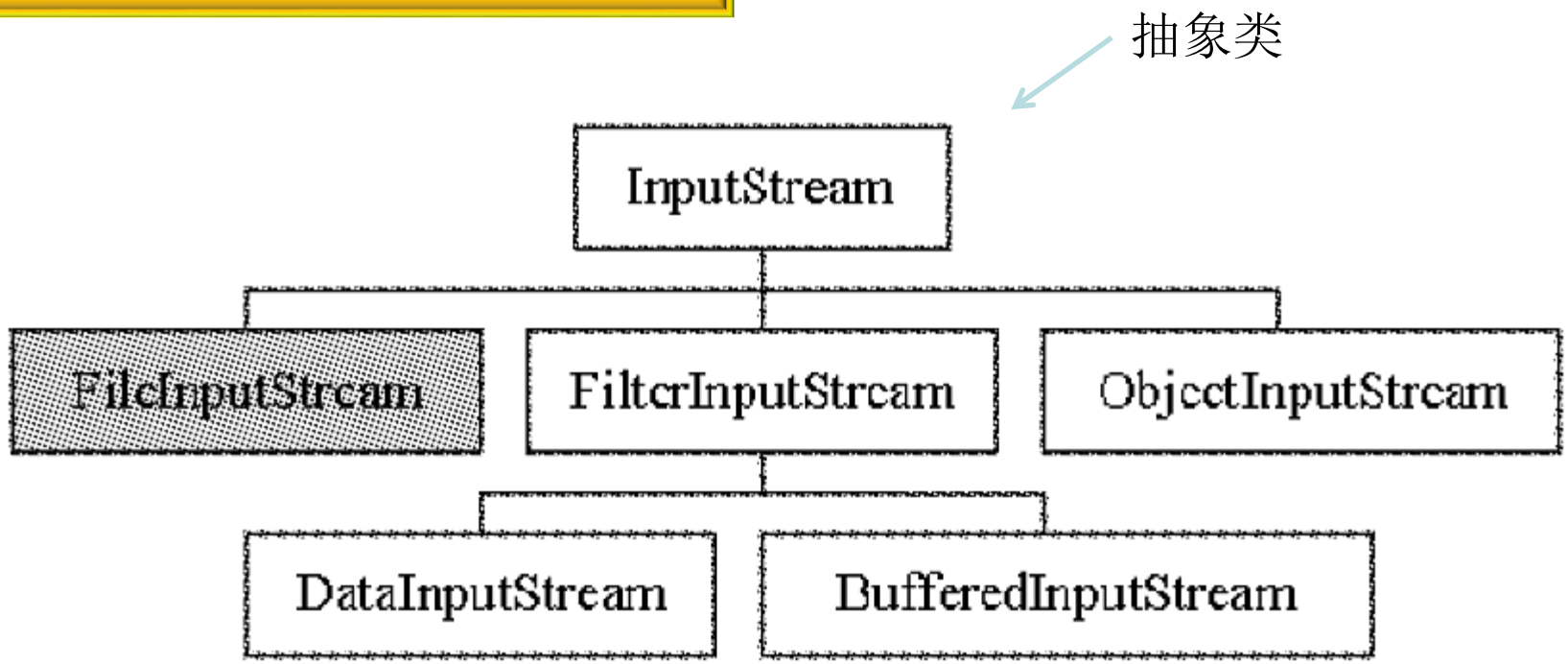
(3) OutputStreamWriter(OutputStream out)：使用系统默认的字符集生成字符输出流。

(4) OutputStreamWriter(OutputStream out, String charsetName)：使用用户指定的字符集生成字符输出流。

看到这里的参数回顾一下

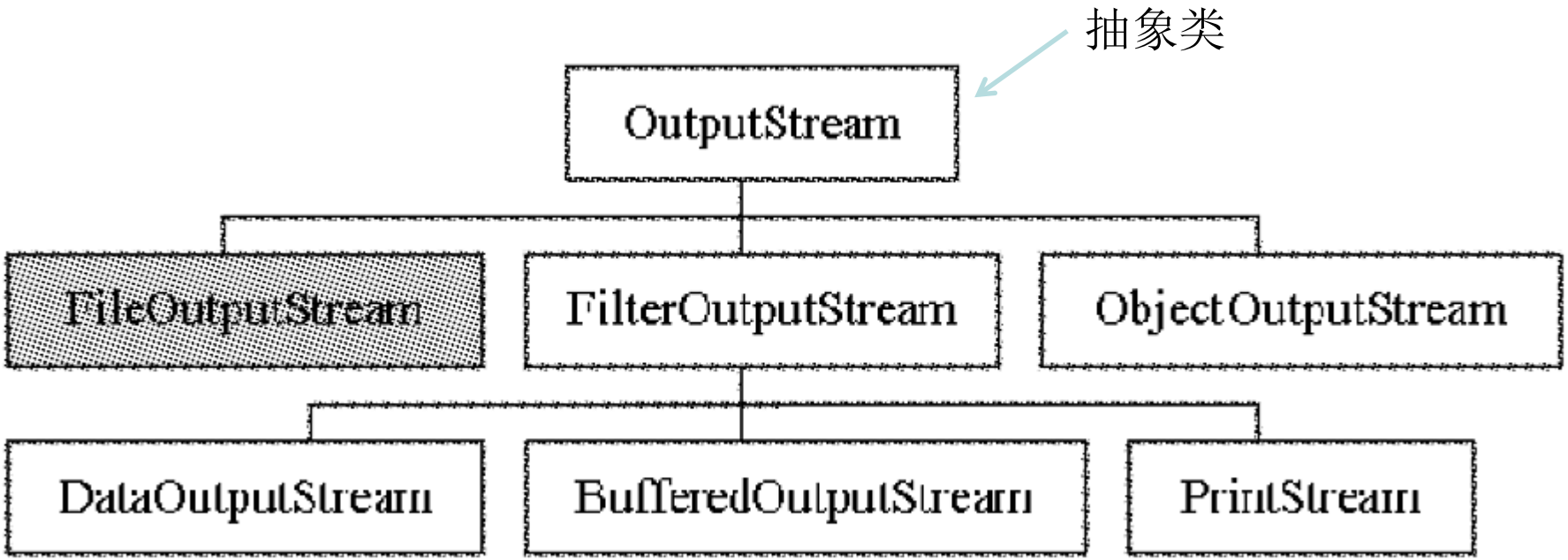
9.3 字节流

9.2.3 Java中的流及其分类



9.3 字节流

9.2.3 Java中的流及其分类



9.4.2 InputStreamReader和OutputStreamWriter类

2.常用方法

除了基类Reader或Wirter定义的方法read()或write()方法外，还包含以下两个方法：

(1) void close()：关闭输入流/输出流。

(2) String getEncoding()：返回转换时所用的字符集。

现在，给出一个这方面的例子：

(1)先创建文件输出流，再用OutputStreamWriter创建字符输出流，之后用几种方式向文件写入一个或多个字符：

(2)创建文件输入流，再用InputStreamReader创建字符输入流，然后读取输入流内容，并显示、输出。

9.4 字符流

9.4.5文件字符流

InputStreamReader或OutStreamWriter的子类的使用：

FileOutputStream fos = new FileOutputStream("char.txt");
OutputStreamWriter osw = new OutputStreamWriter(fos);。

或

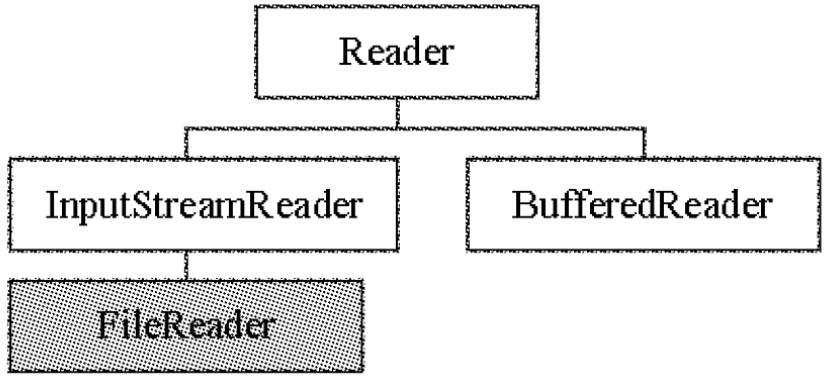
FileInputStream fis = new FileInputStream("char.txt");
InputStreamReader isr = new InputStreamReader(fis)。

ReaderTest.java

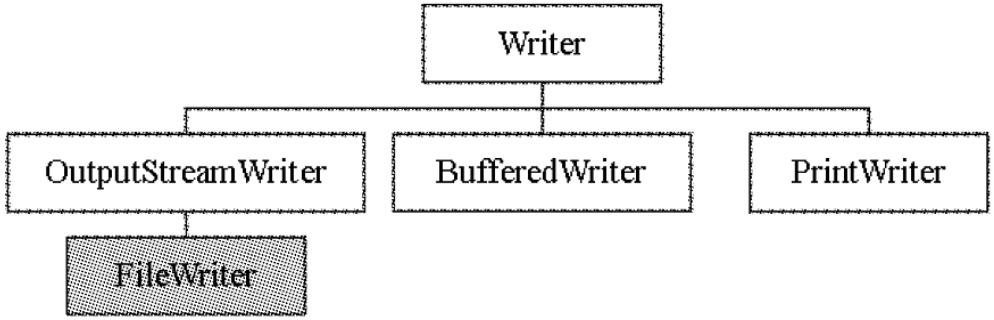
9.4 字符流

9.4.5文件字符流

FileReader和FileWriter是两个系分别如图所示：




```
java.lang.Object
  └java.io.Writer
      └java.io.OutputStreamWriter
          └java.io.FileWriter
```




```
java.lang.Object
  └java.io.Reader
      └java.io.InputStreamReader
          └java.io.FileReader
```

9.4 字符流

9.4.5文件字符流

从图可以看出，这两个类是InputStreamReader或OutStreamWriter的子类，具备从字节流到字符流转换的功能。在前面的例子中，我们分别用了两条语句来实现从字节流到字符流的转换：

FileOutputStream fos = new FileOutputStream("char.txt");

OutputStreamWriter osw = new OutputStreamWriter(fos);。

或

FileInputStream fis = new FileInputStream("char.txt");

InputStreamReader isr = new InputStreamReader(fis)。

有了文件字符流类，我们可以改用下面两条等价语句：

FileWriter fw = new FileWriter("char.txt");

或

FileReader fr = new FileReader(“char.txt”)。

9.4.5文件字符流

FileReader、FileWriter类的构造方法如下：

(1) FileReader(File file)：使用File类型为参数，创建一个FileReader对象。字符集、缓冲区大小使用系统默认设置，下同。

(2) FileReader(String fileName)：使用String类型为参数，创建一个FileReader对象。

(3) FileWriter(File file)：使用File类型为参数，创建一个FileWriter对象。

(4) FileWriter(File file, boolean append)：使用File类型为参数，创建一个FileWriter对象。第二个参数为true将把新增内容追加到文件尾部。

(5) FileWriter(String fileName)：使用String类型为参数，创建一个FileWriter对象。

(6) FileWriter(String fileName, boolean append)：使用String类型为参数，创建一个FileWriter对象。

```
static void testWrite(){
File f=new File("c:\\sayhello.txt");
try{
System.out.println("begin to write");
FileWriter fw=new FileWriter(f);
String str1="hello\n";
String str2="gzx";
fw.write(str1);
fw.write(str2);
fw.close();
System.out.println("end writing");

}
catch(Exception ex)
{
ex.printStackTrace();
}}
```

字符文件输出流

FileReaderTest.java

```
static void testRead(){
File f=new File("c:\\sayhello.txt");
Try{
System.out.println("begin to read");
FileReader fr=new FileReader(f);
char[] tempchs=new char[10];
int i=fr.read(tempchs);
String str=new String(tempchs,0,i);
System.out.println(str);
while(i!=-1){
      i=fr.read(tempchs);
      if (i==-1) break;
            str=new String(tempchs,0,i);
      System.out.println(str);
}
fr.close();
System.out.println("end Reading");
}
catch(Exception ex)
{
ex.printStackTrace();
}

}
```

字符文件输入流

9.3.3 过滤流

思考：

如何用FileInputStream读取一个

int, double, String,……

对于int类型变量a,将其转换为字节数组b

```
int a = 100;
byte[] b = byte[4];
b[3] = (byte)(a & 0xff);
b[2] = (byte)(a>>8 & 0xff);
b[1] = (byte)(a>>16 & 0xff);
b[0] = (byte)(a>>24 & 0xff);
```

字节数组b转换成整形变量a

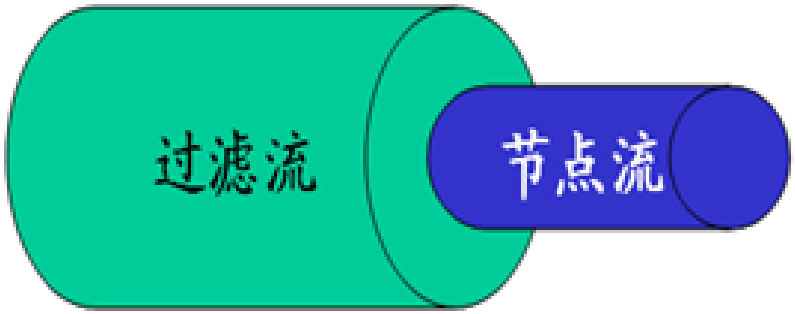
```
int a = 0;
for(int i =0; i< b.length; i++){
    a += (b[i]&0xff) << (24-8*i);
} return a;
```

9.3 过滤流

9.3.3 过滤流

1. “逐层包装”的思想

对于流的构造层数和顺序没有特别要求，只要匹配构造方法的参数类型即可，包装的目的是实现在更高层次上对数据的简便操作。为了说明方便，我们把底层直接与目标设备连接的流称为节点流，而把在节点流之上、对其进行包装的流称为过滤流。



9.3 过滤流

9.3.3 过滤流

1. 字节过滤流

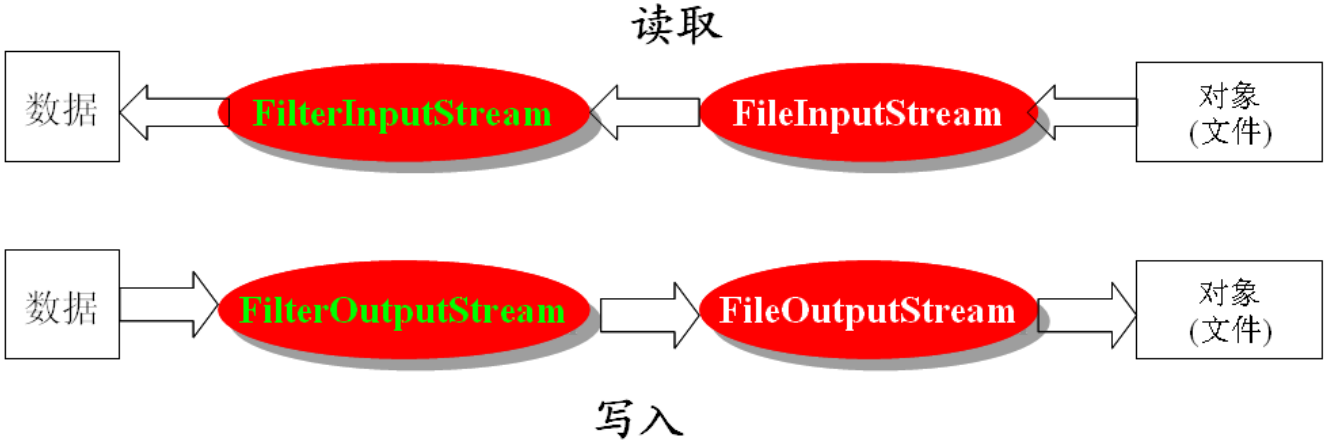
2. 字符过滤流

9.3过滤流

9.3.3 过滤流

2. FilterInputStream与FilterOutputStream类

这两个类都是字节过滤流的基类，它们又派生了多个子类，分别对字节输入流、字节输出流进行特殊处理，在编程时通常不使用这两个基类，而是使用它们的子类。例如：数据流、缓冲字节流都能对字节文件流进行过滤、包装，并能方便地读取各种类型数据、提高读写效率。



字节过滤流　选学

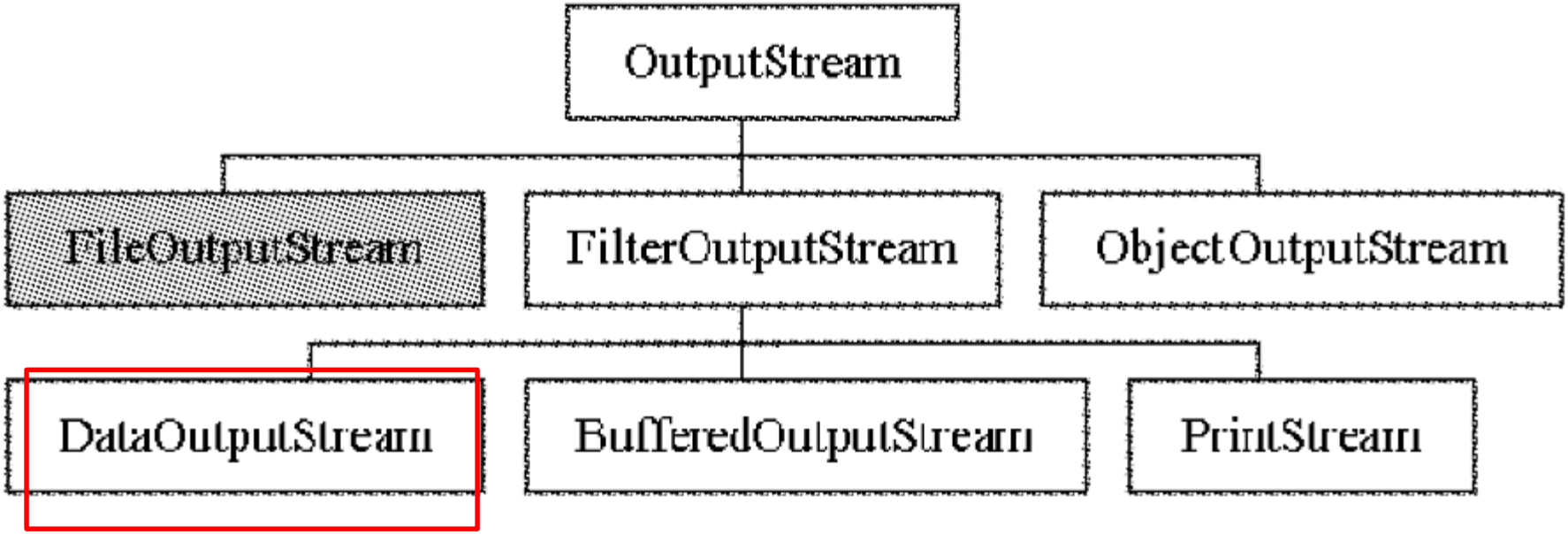
数据流（DataInputStream/DataOutputStream）
字节缓存流（BufferedInputStream/BufferedOutputStream）

可直接转到

字符过滤流

9.3 字节流

9.3.4数据流



9.3 字节流

9.3.4数据流

1.DataInputStream读取基本类型数据和字符串

(1) boolean readBoolean()：从输入流读取一个boolean型

(2) byte readByte()：从输入流读取一个byte型数据。

(3) short readShort()：从输入流读取一个short型数据。

(4) char readChar()：从输入流读取一个char型数据。

(5) int readInt()：从输入流读取一个int型数据。

(6) long readLong()：从输入流读取一个long型数据。

(7) float readFloat()：从输入流读取一个float型数据。

(8) double readDouble()：从输入流读取一个doube型数

(9) String readUTF()：从输入流读取一个String型数据。

boolean	-----
char	16-bit
byte	8-bit
short	16-bit
int	32-bit
long	64-bit
float	32-bit
double	64-bit

9.3 字节流

9.3.4数据流

2.DataOutputStream写入基本类型数据和字符串的方法

(1) void writeBoolean(boolean v)：向输出流写入一个boolean型数据。
(2) void writeByte(int v)：向输出流写入一个byte型数据。
(3) void writeShort(int v)：向输出流写入一个short型数据。
(4) void writeChar(int v)：向输出流写入一个char型数据。
(5) void writeInt(int v)：向输出流写入一个int型数据。
(6) void writeLong(long v)：向输出流写入一个long型数据。
(7) void writeFloat(float v)：向输出流写入一个float型数据。
(8) void writeDouble(double v)：向输出流写入一个doube型数据。
(9) void writeChars(String s)：向输出流写入一个String型数据。
(10) void writeUTF(String str)：向输出流写入一个String型数据。

示例

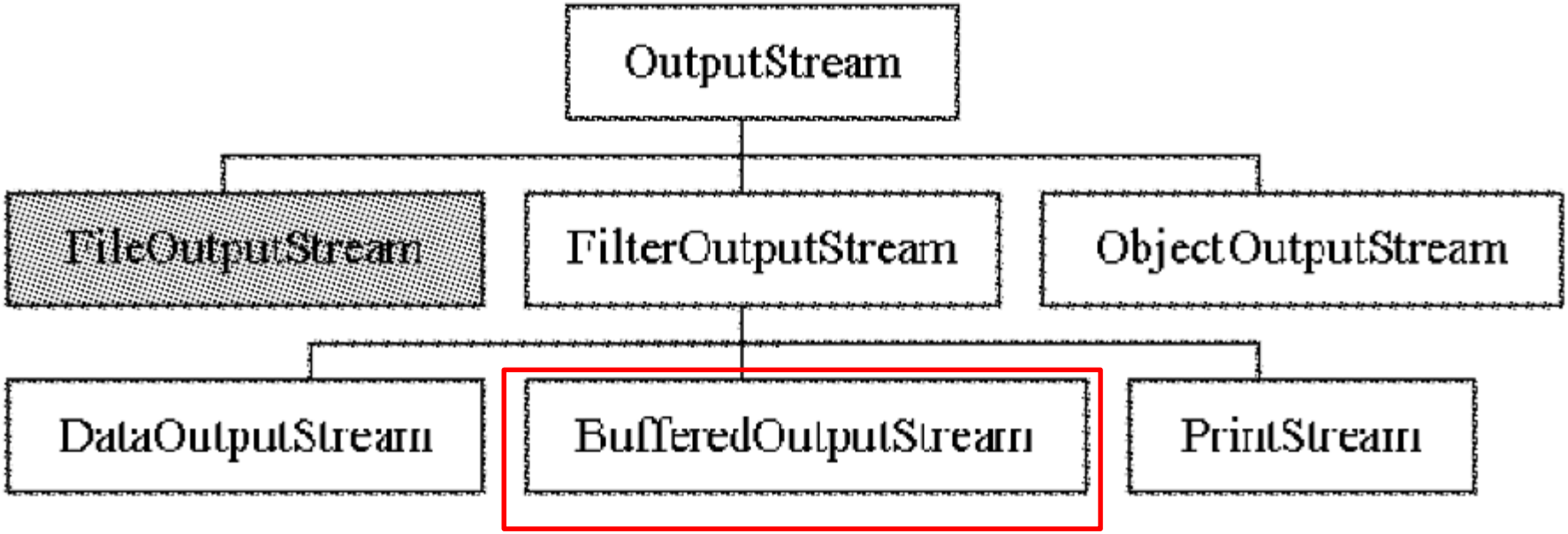
```
FileOutputStream fos=new FileOutputStream("xxx.data");

DataOutputStream dos=new DataOutputStream(fos);
dos.writeInt(100);
dos.writeUTF("DataOutputStream Test");
dos.close();

FileInputStream fis=new FileInputStream("xxx.data");
DataInputStream dis=new DataInputStream(fis);
System.out.println("int:"+dis.readInt());
System.out.println("UTF:"+dis.readUTF());
dis.close();
```

体现了层层包装

```
dos.writeBytes(("DataOutputStream Test”.getBytes());
dos.WriteChars(("DataOutputStream Test”);
```

9.3过滤流

9.3.5缓冲字节流



9.3 字节流

9.3.5缓冲字节流

这是指BufferedInputStream和BufferedOutputStream两个类，请注意不要把类名写错，在Buffer后面带有ed。它们的继承关系分别如图所示。

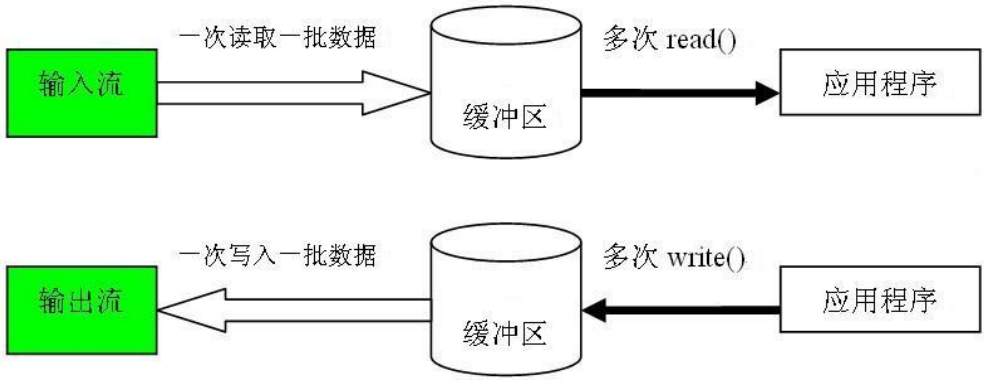
```
java.lang.Object
  └java.io.InputStream
      └java.io.FilterInputStream
          └java.io.BufferedInputStream

java.lang.Object
  └java.io.OutputStream
      └java.io.FilterOutputStream
          └java.io.BufferedOutputStream
```

9.3 字节流

9.3.5缓冲字节流

增加缓冲区有两个基本目的：

1.允许Java程序一次读取，多次操作



2. 允许Java程序一次不只操作一个字节，这样提高了程序的性能。由于有了缓冲区，使得在流上执行skip、mark、和reset方法都成为可能可以加快读写速度，提高存取效率。

9.3 字节流

9.3.5缓冲字节流

BufferedInputStream和BufferedOutputStream的构造方法如下：

(1)BufferedInputStream(InputStreamin)

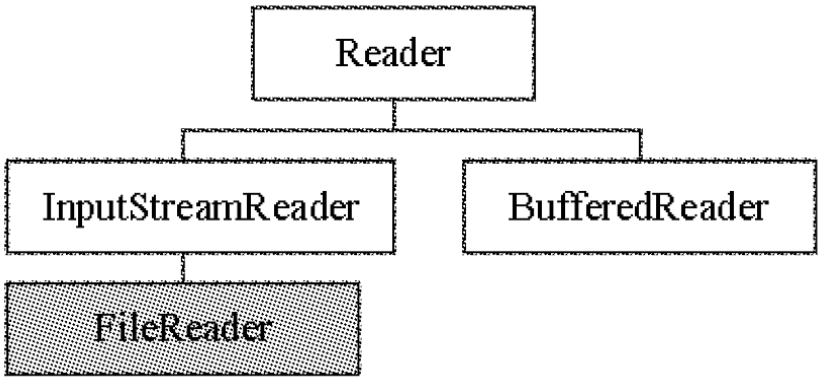
(2)BufferedOutputStream(OutputStreamout)

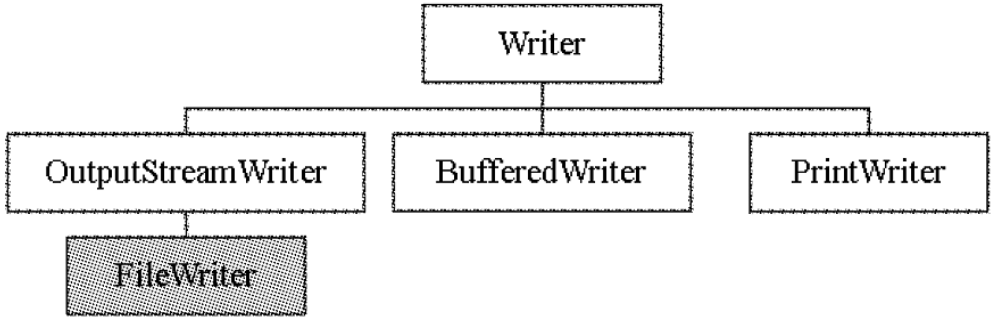
略讲，**重点掌握缓冲字符流**

9.4 字符过滤流

字符流读写的过滤流

-BufferedReader 和BufferedWriter





9.4 字符过滤流

9.4.3 缓冲字符流

与缓冲字节流一样，缓冲字符流由于使用了缓冲区，提高了文本的读写速度。缓冲区的大小可以由用户设置，也可以是系统默认大小(大多数情况下够用)。

BufferedReader和BufferedWirter两个类的构造方法：

(1) BufferedReader(Reader in)：使用系统默认的缓冲区大小生成字符输入流。

(2) BufferedReader(Reader in, int sz)：使用用户指定的缓冲区大小生成字符输入流。

(3) BufferedWriter(Writer out)：使用系统默认的缓冲区大小生成字符输出流。

(4) BufferedWriter(Writer out, int sz)：使用用户指定的缓冲区大小生成字符输出流。

9.4 字符过滤流

9.4.3 缓冲字符流

这两个类经常使用的原因是，提供了字符读写的便利方法：

(1)缓冲字符输入流类提供了一个“整行字符读取”方法：

格式： String readLine()

功能：能够整行地读取字符，遇到换行符为止(注意：不同操作系统的换行符不同，例如：Windows系统是“\r\n”(即回车换行符)，Linux系统是”\n”(即换行符))。当无数据可读时，将返回null，可以此来判断数据是否读取完毕

(2)缓冲字符输出流提供了一个“换行符”方法，

格式：void newLine() 功能：能够根据不同的操作系统，提供相应“换行符”

9.5 对象序列化

9.5.1 对象序列化的概念

1.概念

序列化：把Java对象转换为字节序列的过程。

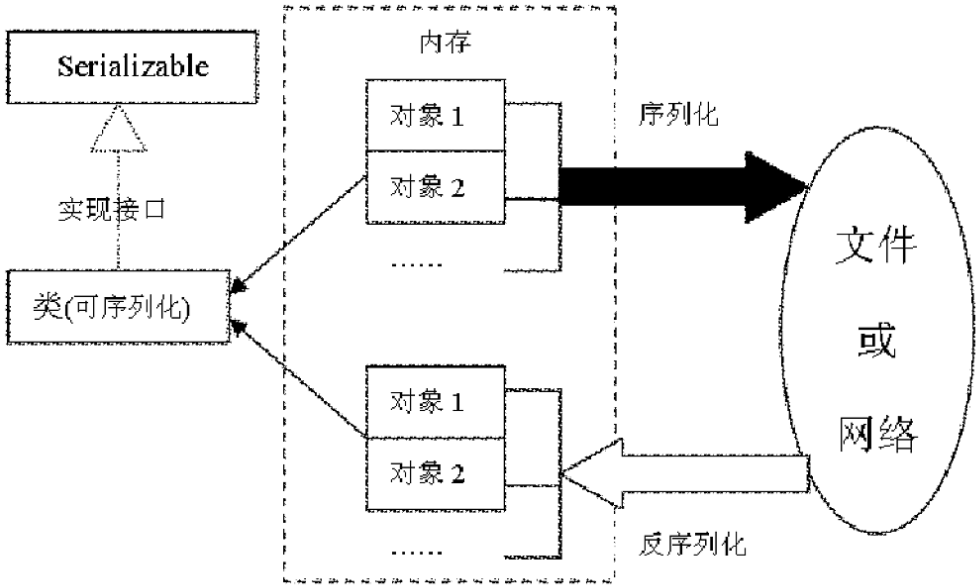
反序列化：把字节序列恢复为Java对象的过程。

2.用途

对象的序列化主要有两种用途：

1） 把对象的字节序列永久地

2） 在网络上传送对象的字节



9.5 对象序列化

9.5.1 对象序列化的概念

对象要具备序列化的功能，必须实现java.io.Serializable接口，该接口是一个空接口，不包含任何方法，又称**标记接口**，String、Date等实现了此接口。

ObjectOutputStream代表对象输出流，它的writeObject(Object obj)方法可对参数指定的obj对象进行序列化，把得到的字节序列写到一个目标输出流中。**只有实现了Serializable和Externalizable接口的类的对象才能被序列化**。

ObjectInputStream代表对象输入流，它的readObject()方法从一个源输入流中读取字节序列，再把它们反序列化为一个对象，并将其返回。

9.5 对象序列化

9.5.2 ObjectInputStream和ObjectOutputStream类

1．构造方法：

(1)ObjectInputStream(InputStream in)：以字节输入流为参数，创建对象输入流

(2)ObjectOutputStream(OutputStream out)：以字节输出流为参数，创建对象输出流

```
ObjectOutputStream out = new ObjectOutputStream(new
FileOutputStream("D:\\objectFile.obj"));

ObjectInputStream in = new ObjectInputStream(new
FileInputStream("D:\\objectFile.obj"));
```


2．主要读写方法：

除继承基类的read()/write()方法外，还提供了读/写各类数据的readXxx()/writeXxx()方法，特别是对象的读写方法：

(1)Object readObject()：从对象输入流中读取对象

(2)writeObject(Object obj)：向对象输出流写入对象

```
out.writeObject("你好!"); //写入字面值常量
out.writeObject(new Date()); //写入匿名Date对象
Customer customer = new Customer(“张三", 24);
out.writeObject(customer); //写入customer对象
out.close();
```

2．主要读写方法：

除继承基类的read()/write()方法外，还提供了读/写各类数据的readXxx()/writeXxx()方法，特别是对象的读写方法：

(1)Object readObject()：从对象输入流中读取对象

(2)writeObject(Object obj)：向对象输出流写入对象

```
String tempstr=(String) in.readObject()); //读取字面值常量
Date dt=(Date) in.readObject()); //读取匿名Date对象
Customer obj = (Customer) in.readObject(); //读取customer对象
in.close();
```

对象的序列化和反序列化

- 多个自定义对象的序列化和反序列化：

```
Employee har=new Empleyee(“Ha”,5000,1990,12,1);
Manager bos=new Manager(“Cal”,80000,1988,2,13);
ArrayList lst=new ArrayList();
lst.add(har);
lst.add(bos);
out.writeObject(lst);
```

  - 使用对象输入流的readObject方法按次序读取对象

```
ArrayList lst=(ArrayList)in.readObject();
Employee e1=(Employee)lst.get(0);
Manager e2=(Manager)lst.get(1);
```

对象的序列化和反序列化

- 关于序列化/反序列化的说明：通常，对象中的所有属性都会被序列化,但要注意：
  1. 静态成员: 不能被序列化,因为 静态在方法区里面 ,不再 堆里面,
  2. transient: 修饰成员变量, 使它不被序列化
     - 有些敏感信息（例如密码），一旦被序列化，就可以通过文件或网络拦截进行偷窥——出于安全的考虑，需要对某些属性禁止序列化，对这类属性使用transient修饰

对象的序列化和反序列化

- 若要把禁止序列化的成员序列化，
  - 一个方法是去除transient修饰符，
  - 另一方法是在需要序列化的类中重写writeObject方法
    - 序列化时，若该类有字节的writeObject方法，则对象输出流调用该对象自己的writeObject方法，否则调用默认的
    - 相应的需要设计对应的readObject方法

```
class Book implements Serializable {
    String bookname;
    String author;
    transient double price;
    Book(String bookname,String author,double price){
        this.bookname=bookname;
        this.author=author;
        this.price=price;
    }
    public String toString(){
        return "bookname:"+bookname+" author:"+author+" price:"+price;
    }
}
```

```
private void writeObject(ObjectOutputStream out) throws
    IOException {
        out.defaultWriteObject();
        out.writeDouble(price);
    }
private void readObject(ObjectInputStream in) throws
    IOException, ClassNotFoundException {
        in.defaultReadObject();
        price = in.readDouble();
    }
```

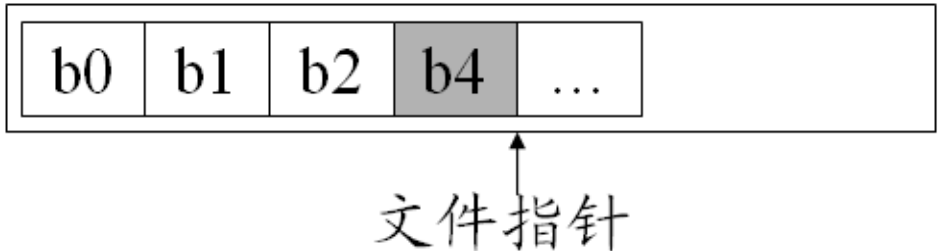
注意：这里的方法必须是private类型，若是public的不能重载到。

- 还需注意：

  ObjectInputStream使用时需要保证**对象所对应的类**要在程序的CLASSPATH中

随机访问文件类 RandomAccessFile

- 关于流：
  - InputStream/OutputStream，或者Reader/Writer，它们的共同特点是：
    - 只能按照数据的先后顺序读取数据源的数据
    - 只能按照数据的先后顺序向数据汇写数据
    - 输入流和输出流各司其职
- RandomAccessFile类：
  - 不属于流，具有随机读写文件的功能，能从文件的任意位置开始执行读写操作
  - 利用它打开文件时，**既可以“读取”也可以“写入”**，还可根据需要任意“拨动”文件读写指针，



随机访问文件类 RandomAccessFile

- RandomAccessFile类：
  - 构造方法：
    - RandomAccessFile(File file, String mode)
    - RandomAccessFile(String name, String mode)
    - 说明：mode取值可以是：r(只读)；rw(读写)——注意，该类不支持只写模式，所以w是非法的
  - 该类提供了用于定位文件位置的方法：
    - getFilePointer()：返回当前读写指针所处的位置
    - seek(long pos)：设定读写指针的位置，与文件开头相隔pos个字节
    - skipBytes(int n)：从当前位置跳过n个字节
    - length()：返回文件包含的字节数

随机访问文件类 RandomAccessFile

```
1  import java.io.*;
2  public class RandomTester{
3     public static void main(String[] args){
4        RandomAccessFile rf=new RandomAccessFile("g:\\test.data","rw");
5        for (int i=0;i<10;i++)
6           rf.writeLong(i*1000);
7
8        rf.seek(5*8);  //从文件开头跳过5个long数据
9        rf.writeLong(1234);
10
11       rf.seek(0);     //把读写指针定位到文件开头
12       for (int i=0;i<10;i++)
13          System.out.println("Value "+i+": "+rf.readLong());
14
15       rf.close();
16    }
17 }
```

boolean	-----
char	16-b
byte	8-bit
short	16-b
int	32-b
long	64-b
float	32-b
double	64-b

随机访问文件类RandomAccessFile

- 小结
  - 工作方式把DataInputStream和DataOutputStream结合起来，再加上它自己的一些方法，比如定位用的getFilePointer( )，在文件里移动用的seek( )，以及判断文件大小的length( )、skipBytes()跳过多少字节数。
  - 在读取时必须已经知道该文件的存储结构，比如第一个是Integer类型，第二个是Double类型等，否则可能定位不准确从而读出乱码

本章小结

本章介绍的“文件与输入输出流”是Java的重要内容，凡是有输入输出操作的地方，都要用到相关知识点。而熟练地掌握它们并非易事，仅所涉及的类与接口就有几十个，需要理解一些基本概念和重要思想，理顺类与类之间的相互关系。

File类是以抽象方式表示文件和目录，通过该类对象可查看对应文件与目录的基本信息，进行创建、删除、改名等操作，但不涉及文件内容的读写，所表示的文件、目录可能存在，也可能不存在。

本章小结

Java的输入输出操作大多是以“流”方式来进行的。根据方向的不同，“流”可分为输入流、输出流两种类型，需要注意的是：应始终站在内存(即应用程序)的角度来区分是输入流还是输出流。“流”具有明确分工，只能从输入流中读取数据、向输出流写入数据，执行相反操作就会出错。如果按照数据处理基本单位的不同，Java中的“流”又可分为字节流、字符流两种类型，字节流以8位字节为处理单位，可操作各种类型数据；字符流的处理单位是16位的Unicode编码字符，适合进行文本操作。若将上述两种不同标准的划分组合起来，就能形成4种基本流，即：字节输入流、字节输出流、字符输入流、字符输出流。正确分辨各种类型的流，是进行I/O操作的前提条件。

本章小结

字节流的两个基类分别是InputStream、OutputStream，它们对应的子类命名格式分别为XxxxInputStream、XxxxOutputStream(Xxxx是子类名前缀)，字节输入流的主要操作是“读取”数据，对应的基本方法是read()，字节输出流的主要操作是“写入”数据，对应的基本方法是write()。字节流中的重要类是FileInputStream和FileOutputStream，它们提供了读写文件的基本方法，但用来操作字符串和其它类型数据并不方便，这时，可以请“过滤流”来帮忙。“过滤流”体现了“逐层包装”思想，即用一个已存在的流来构造另一个流，构造的目的是让操作更方便。“数据流”(DataInputStream和DataOutputStream)即是这方面的典型代表，它们分别实现了DataInput和DataOutput接口，适合操作各种类型数据。缓冲字节流(BufferedInputStream和BufferedOutputStream)由于引入缓冲区，大大加快了数据的读写速度。PrintStream类提供了print()和println()等方法，能够以字符串方式输出各种类型的数据，常见的System.out、System.err都是它的实例(System.in为InputStream类型)。

本章小结

字符流的两个基类分别是Reader、Writer，与字节流类似，它们对应的子类命名格式分别为XxxxReader、XxxxWriter(Xxxx是子类名前缀)，字符输入流和字符输出流的主要操作分别是“读取”和“写入”数据，对应的方法也是read()和write()，但操作对象不同。从名字可以看出，InputStreamReader和OutputStreamWriter是实现字节流与字符流转换的桥梁，它们是将字节流包装成字符流的基础。与缓冲字节流一样，缓冲字符流(BufferedReader和BufferedWirter)借助缓冲区，提高了文本的读写速度，并提供了“整行”读取字符串、自动添加“换行符”等方法。文件字符流实现了“文件字节流+字节/字符转换流”的组合功能。从JDK 1.5起，PrintWriter类的功能得到了显著增强，用它可以取代文件字符流、缓冲字符流类，在输出方面有很多优点。

本章小结

对象的序列化/反序列化是实现对象“整存整取”的有效方法，存放位置既可以是本地文件，也可以是网络。对象要具备序列化的功能，必须实现Serializable接口，该接口是一个空接口，不包含任何方法。ObjectInputStream、ObjectOutputStream类分别提供了readObject()、writeObject()方法来读、写对象。

RandomAccessFile类具备随机存取文件的功能，该类不同于前面介绍的输入流、输出流类，利用它打开文件时，既可以进行“读取”操作，也可以进行“写入”操作，还可根据需要任意“拨动”文件读写指针。

本章小结

本章重点：File类，输入输出流，字节流，字符流，对象序列化，RandomAccessFile类；难点：输入流与输出流、字节流与字符流的区分，类与类之间的相互关系，“逐层包装”思想的理解。