1、什么是File

file对象可以用于封装磁盘上一个文件或目录的路径，
如果路径对应的文件或目录在磁盘上存在，呢么就可以
通过File对象来对磁盘上文件或目录进行操作了

注意：

1>File对象既可以封装文件路径，也可以封装目录路径

2>File对象封装的路径在磁盘上不一定有对应的文件或目录

3>File对象是在内存中的，文件或目录是在磁盘中的。

2、File对象创建

1>File f = new File("文件目录或路径");

2>File f = new File("父目录路径","子文件或子目录名称");

3>File f = new File("父目录","子文件或子目录名称");

3、主要方法

1>exists() 判断文件或目录是否存在

2>isFile() 判断是否为文件

3>isDirectory 判断是否为目录

4>delete() 删除文件或空目录

5>createNewFile 创建文件

6>mkdir() 创建单机目录

7>mkdirs() 创建多机目录

8>getParentFile() 获取父目录

9>getName() 获取文件或目录名称

10>length() 获取文件大小

11>getAbsolutePath 获取文件或目录绝对路径

12>lastModified 获取文件最后修改时间

13>setLastModified 修改文件最后修改时间

14>list() 列出目录下所有子文件和子目录名称

15>listFiles() 列出目录下所有子文件和子目录

16>listFiles(FilenameFilter filter) 根据文件名过滤子文件或子目录

17>listFiles(FileFilter filter) 根据文件属性过滤子文件或子目录

一、IO流

I：in输入

O：out输出

流：stream 数据的流动

通常我们编写的程序，除了自身会定义一些数据除外。经常需要从外界来读取一些数据，或者将程序中产生的一些新的数据发送到外界进行永久存储（内断电数据就会丢失），这个时候我们就需要用到IO流

I：输入流，程序从外界读取数据

O：输出流，程序向外界写出数据

程序都是运行在内存中的

外界一般是磁盘文件，数据库，网络

二、IO流的分类

1、按照数据的流动方法分

1>输入流

程序从外界读取数据的流管道

2>输出流

程序将数据写出到外界的流管道

2、按照流管道传输的数据类型分

1>字节流

逐个字节传输数据，可以传输一切类型的文件数据

2>字符流

逐个字符传输数据，只能传输纯文本文件数据，如 .txt 或 .java

不能传输二进制文件，如：图片，音频，视频.....

3、按流管道的功能来分

1>节点流(低级流)

直接与数据源头相接，接文件或接程序

2>处理流(高级流)

套接着其他节点流，用于增强功能

三、字节流

抽象父类：InputStream/OutputStream

InputStream:字节输入流抽象父类

OutputStream:字节输出流抽象父类

1、FileInputStream/FileOutputStream

文件字节输入流/文件字节输出流

1>功能：节点流，可以从磁盘文件中读取数据

输出流可以向磁盘文件中写出数据

2>对象创建

输入流：

FileInputStream in = new FileInputStream("文件路径");

FileInputStream in = new FileInputStream("文件对象");

输出流：

FileOutputStream out =

new FileOutputStream("文件路径" | 文件对象)

FileOutputStream out =

new FileOutputStream("文件路径" | 文件对象,boolean append)

注意：

1）输入流要求文件必须存在，否则会抛出异常。

2）输出流文件不存在会自动创建，但要求文件的父目录必须要存在。

3）输出流append参数为true表示追加，没有这个参数或为false时表示覆盖。

3>主要方法

3.1>输入流

1)int read()

每次调用从文件中读取一个字节，读到内存后前面再补上三个字节，

也就是24个0，以int形式返回，读到文件末尾返回-1

2)int read(byte[] buf)

每次调用按照buf字节数组的最大长度去文件中读取一批字节，

然后存入到buf字节数组，返回本次实际上读取到的字节个数，

读到文件末尾返回-1

3>int read(byte[] buf,int offset,int length)

每次调用去文件中最大读取length个字节，然后将读取到的数据

往buf字节数组的第offset个下标位置开始存储，

返回本次实际上读取的字节个数，读到文件末尾返回-1

3.2>输出流

1)void write(int b)

将int类型参数b的最后1个字节写出到文件中

而忽略掉前面三个字节

2)void write(byte[] data)

一次性将字节数组data中所有数据都写出到文件中

3)void write(byte[] data,int offset,int length)

将data字节数组中第offset个下标位置开始的length个

字节写出到文件中

3.3>关闭流

close() 用于关闭输入流和输出流

复制

单字节

多字节

2、BufferedInputStream/BufferedOutputStream

字节缓存输入流/字节缓冲输出流

1>功能：缓冲流，要接字节流，内部自带一个8192个长度的字节数组，

用于提升读写效率(每次读8k)

2>对象创建

2.1>输入流

```
BufferedInputStream in =
        new BufferedInputStream(
            new FileInputStream(字节输入流)
        );
```

2.2>输出流

```
BufferedOutputStream out =
        new BufferedOutputStream(
            new FileOutputStream(字节输出流)
        );
```

3>主要方法

3.1>输入流

int read()

每次通过内部自带的缓冲区读取一批字节数据。

3.2>输出流

1)void write(int b)

将字节放入到缓冲区中，然后再将字节数组写出到流中。

2)void flush()

如果缓冲流内部自带的字节数组数据没有满，呢么是不会写出到流中的，
通过手动调用flush方法将字节数组中所有数据冲刷到流中。
或者调用close()方法，这个方法内部也会调用flush()方法。

3、ObjectInputStream/ObjectOutputStream

对象字节输入流/对象字节输出流

1>功能：序列化流,能将一个对象转为字节序列写出到流中

也能从流中读取字节序列转为一个对象

2>对象创建

2.1>输入流

ObjectInputStream ois = new ObjectInputStream(字节输入流);

2.2>输出流

ObjectOutputStream oos = new ObjectOutputStream(字节输出流);

3>主要方法

3.1>void writeObject()

将一个对象转为字节序列写出到流中

3.2>Object readObject()

从流中读取字节序列转为一个对象

4>序列化和反序列化

序列化:将内存中一个对象拆散成一堆字节的过程,

要求对象的类型上必须要实现Serializable接口

反序列化:将一堆字节合成为内存中一个对象的过程

序列化和反序列化的目的都是为了对象可以再流中或网络中进行传输.

注意：类中使用static修饰的静态变量和

使用transient修饰的实例变量不能被序列化

4、DataInputStream/DataOutputStream

数据固定字节输入流/数据固定字节输出流

1>功能：主要用于读写基本类型和字符串类型

2>对象创建

2.1>输入流

DataInputStream in = new DataInputStream(字节输入流)

2.2>输出流

DataOutputStream out = new DataOutputStream(字节输出流)

3>主要方法

3.1>输入流

1)readXXX() XXX:代表8种基本类型

2)readUTF() 读取一个字符串

3.2>输出流

1)writeXXX() XXX:代笔8种基本类型

2)writeUTF() 以UTF-8编码格式写出字符串

5、PrintStream

打印流（输出流）

1>主要功能：可以将任意数据类型转为字符串进行打印输出。

2>对象创建

PrintStream ps = new

PrintStream(文件路径||文件名||自带刷新||编码方式||字节输出流);

3>主要方法

1>print() 不换行打印

2>println() 换行打印

4、字符流

主要用于读写纯文本类型文件（如:txt.java）

超文本标记语言 html

图片/音频/视频/doc等二进制文件只能用字节流，不能用字符流

抽象父类

Reader:字符输入流抽象父类

Writer:字符输出流抽象父类

1、FileReader/FileWriter

文件字符输入流/文件字符输出流

1>功能：节点流，可以从流中读取字符，也可以向流中写出字符

2>对象创建：

2.1>输入流

FileReader reader = new FileReader("文件路径"|文件对象);

2.2>输出流

FileWriter writer = new FileWriter(""文件路径"|文件对象");

FileWriter writer =

new FileWriter(""文件路径"|文件对象,append是否追加");

注意:

1)输入流要求文件必须存在，否则会抛出异常。

2)输出流文件不存在会自动创建，但是要求文件的父目录必须存在。

3)输出流append参数为true表示追加，没有这个参数或参数为false表示覆盖。

3>主要方法

3.1>输入流

1)int read()

从流中读取一个字符到内存，然后在前面再补上2个字节16个0，以int形式返回，读到文件末尾返回-1。

2)int read(char[] buf)

按照buf字符数组最大长度每次去流中读取一批字符，然后返回本次实际上读取的字符个数，读到文件末尾返回-1。

3)int read(char[] buf,int offset,int length)

每次去流中最多读取length个字符，然后将读到的字符数据往buf字符数组中第offset个下标文职开始存，返回本次实际上读取的字符个数，读到文件末尾返回-1。

3.2>输出流

1)write(int c)

将参数c的最后2个字节写出到流中，忽略前面2个字节。

2)write(char[] buf)

将buf字符数组中所有数据写出到文件中。

3)write(char[] buf,int offset,int length)

将buf字符数组第offset个下标文职开始的length个字符写出到文件中。

4)write(String str)

等价于write(char[] buf)

5)write(String str,int offset,int length)

等价于write(char[] buf,int offset,int length)。

6)flush()

刷新缓冲区。

2、InputStreamReader/OutputStreamWriter

编码转换输入流/编码转换输出流

1>功能：编码转换流，可以按照指定的编码方式

将字节流转换为字符流。

2>对象创建

2.1>输入流

InputStreamReader in =

new InputStreamReader(字节输入流,"编码方式")

2.1>输出流

OutputStreamWriter out =

new OutputStreamWriter(字节输出流,"编码方式")

3>主要方法

继承自抽象类的方法

3、BufferedReader/BufferedWriter

缓冲字符输入流/缓冲字符输出流

1>功能：缓冲流，内置一个8192长度的字符数组

用于提升读写效率

2>对象创建

2.1>输入流

BufferedReader in = new BufferedReader(字符输入流);

2.2>输出流

BufferedWriter out = new BufferedWriter(字符输出流);

3>主要方法

3.1>输入流

1)String readLine()

每次读取一行数据，读到文件末尾返回null

2)输出流

1)newLine()

换行

4>PrintWriter

字符打印流，作用与用法与printStream相同

1>功能：可以将任意数据类型转为字符串进行打印输出

2>对象创建

PrintWrite out = new PrintWriter(文件路径|文件对象|自动刷新|

编码方式|字节输出流|字符输出流);

3>主要方法

与PrintStream相同

五、Java.io.RandomAccessFile(了解)

随机访问文件，将文件当作是一个大的字节数组，

通过文件指针（数组下标）可以实现对文件的任意位置访问

这个类不属于ＩＯ流范畴，但是拥有和ＩＯ流相同的读写功能

1、对象创建

RandomAccessFile raf =

new RandomAccessFile("文件路径"|文件对象,"rw"|"r");

2、主要方法

1)read()

2)readXX()

3)write()

4)writeXXX()

5)getFilePointer() 获取文件指针

6)seek() 移动文件指针

对象刚创建时，文件指针默认指向0位置，每写出或者读取一个字节后，文件指针将会自动向后移动一个位置。

文件拆分