**JavaIO简介：**

JavaIO指的是Java的输入输出。在Java中，所有与输入输出操作有关的类都定义在java.io 包中。

**File类：**

File类表示文件操作类，使用File类可以创建文件或删除文件等操作。

**File类的分隔符：**

**File类常用操作（方法）：**

1. 使用createNewFile( )方法可以创建一个文件。创建文件时，如果文件存在，则不会 创建。
2. 使用delete( )方法可以删除一个文件，但在删除一个文件时最好使用exists( )方法判 断文件是否存在。
3. 使用mkdir( )方法可以创建一个文件夹，但在创建一个文件夹时应该使用 getParentFile( )方法得到父目录路径并使用exists( )方法判断其是否存在，如果父目录 存在，则创建文件夹。

4）使用isDirectory( )方法可以判断给定的一个路径是否是目录。

5）当File类对象是一个目录的话，则可以使用list( )方法或listFiles( )方法得到目录中的 文件名或路径。

·public String[] list( )：列出全部名称，返回一个字符串数组。

·public File[] listFiles( )：列出完整路径，返回一个File类数组。

**RandomAccessFile类：**

RandomAccessFile表示随机读写类，可以用来进行文件的读写操作，并且，在读取数据 可以移动指针来控制读取的位置。

但过于复杂，优点是能够移动指针控制读取的位置。

**流的概念：**

1）在程序中，所有的数据都是以流的方式进行传输和保存的。使用输入流读取数据， 使用输出流写入和保存数据。

2）在Java中，流分为字节流和字符流两种，每种流都有自己的输入输出操作。在字节 流中使用InputStream类完成输入操作，使用OutputStream类完成输出操作。在字符 流中，使用Writer类完成输出操作，使用Reader类完成输入操作。

**Java中IO操作思路：**

1. 声明File类，打开一个文件；
2. 通过字符流或字节流完成输入或输出操作并指定位置。
3. 进行读/写操作
4. 关闭输入/输出流。

**字节流：**

字节流主要操作byte类型的数据，以byte类型的数组为准，主要操作类是InputStream 类和OutputStreaml类。这些类都是抽象类，根据不同子类实例化其输入或输出的位置 也不同。

**FileOutputStream类：**

FileOutputStream类表示文件输出，是OutputStream类的子类，通过FileOutputStream 类实例化的OutputStream类对象就具备了向文件中写入数据的功能。

1）实例化FileOutputStream类对象时需要传入一个File类对象，如果文件不存在，则 会自动创建。

2）如果指定的文件中有数据，则执行程序后，原文件中的数据会被覆盖；如果不想被 覆盖，则在使用FileOutputStream类实例化OutputStream类对象时在构造方法中加 入ture；表示向文件中追加内容。

3）在向文件中写入数据时，如果想要在文件中进行换行，则需要再写入的数据时加入

换行符”\r\n”。

**FileInputStream类：**

FileInputStream类表示文件输入，是InputStream类的子类，通过FileInputStream类实 例化的InputStream类对象就具备了从文件中读取数据的功能。

1. 实例化FileInputStream类对象时需要制定File类对象，如果指定的文件不存在，则 会出现异常；
2. 如果不知道要读取的文件中数据有多大，则可以通过判断是否读到文件末尾的方式 来读取文件。如果读到文件末尾了，则返回-1；
3. 在File类中存在一个length( )方法，此方法返回文件的大小，这样就可以避免开辟 过多的无用空间。
4. 通过read( )方法的返回值可以知道从文件中读了多少数据。

**字符流：**  
 字符流操作操作的类型主要是String类，主要操作类是Writer类和Reader类。这些类 都是抽象类，根据不同子类实例化其输入或输出的位置也不同。字符流的唯一好处是可 以直接操纵字符串。

**FileWriter类：**

FileWriter类表示文件输出，是Writer类的间接子类。通过FileWriter类实例化的Writer 类对象就具备了向文件写入数据的功能。

1. FileWriter类实例化时，需要指定要操作的文件；如果指定的文件中有数据，则执行 程序后，原文件中的数据会被覆盖；如果不想被覆盖，则在使用FileWriter 类实例 化Writer类对象时在构造方法中加入ture；表示向文件中追加内容。
2. 使用Writer类向文件中写入数据完毕后，一定要关闭输出流，否则将写入失败。除 非调用了flush( )方法；

**FileReader类：**

FileReader类表示文件输入，是Reader类的间接子类。通过FileReader类实例化的Reader 类对象就具备了从文件中读取数据的功能。

1. FileReader类实例化时，需要指定要操作的文件；
2. 如果不知道要读取的数据的长度，也可以通过循环的方式来判断是否读到文件末尾 了；或者通过File类中length( )方法开辟相应大小的char数组，然后将数据一次性 读取到char数组中；在或者通过read( )方法返回值来判断读取了多少数据。

**字节流与字符流的区别：**

字节流是对文件本身进行操作的，而字符流在操作时使用了缓冲区，通过缓冲区再对文 件进行操作。具体的表现为：

使用字节输出流向文件中写入数据后，如果没有关闭输出流，内容也会写进成功；但使 用字符输出流向文件中写入数据后，如果忘记关闭输出流，则数据不会写入到文件中， 除非使用flush( )方法强制清空缓冲区。

在开发中，使用字节流更好。

**文件复制的原理：**

文件复制的原理是通过输入流一边读取数据，一边将读取的数据使用输出流写入文件中。

**转换流：** 转换流指的是OutputStreamWriter类和InputStreamReader类，它们的功能是实现字符 流与字节流之间的转换。

1. OutputStreamWriter类是Writer类的子类，功能是将输出的字符流转变为字节流。
2. InputStreamReader类是Reader类的子类，功能是将输入的字节流转变为字符流。

转换流是为字符流（Writer类和Reader类）使用的。

**关于转换流继承结构的说明：**

从JDK文档可以知道，FileOutputStream是OutputStream类的直接子类，FileInputStream 类是InputStream类的直接子类，但在字符流中，FileWriter类并不是Writer类的直接子 类，而是OutputStreamWriter类的直接子类，FileReader类也不是，Reader类的直接子 类。但OutputStreamWriter类和InputStreamReader类分别是Writer类和Reader类的直 接子类。所以可知，无论是使用字节流还是字符流，最终都是以字节的形式进行输入输 出操作的。

**内存操作流：**

假如某种操作要进行IO操作，但是又不希望在磁盘上产生文件，此时可以将内存作为 一个临时文件进行操作，只需将信息写入内存流中。在Java中，内存操作流有两大类：

1. 字节内存流：ByteArrayInputStream类（内存字节输入流）、ByteArrayOutputStream （内存字节输出流）
2. 字符内存流：CharArrayInputStream类（内存字符输入流）、CharArrayOutputStream （内存字符输出流）

内存操作流一般在生成一些临时文件时才会使用。

**管道流：** 1）管道流又称线程通信流，主要作用是进行两个线程之间的通信。

2）管道流分为管道输出流（PipedOutputStream类）和管道输入流（PipedInputStream 类）。

3）如果想要进行管道输出，则必须把管道输出流连在输入流上，可以使用管道输出流 （PipedOutputStream类）中的connection( )方法进行两个管道流的连接。启动线程后 就会自动的进行管道的输入、输出操作。

**打印流：** 打印流是输出流的一种，是输出操作最方便的类。打印流可以打印任何的数据类型。打 印流分为字节打印流（PrintStream类）和字符打印流（PrintWriter类）。主要是字节打 印流为主。

PrintStream类是OutputStream类的子类。

**装饰者设计模式：** 以打印流进行讲解。打印流PrintStream类是OutputStream类的子类，但是，PrintStream 类有一个方法是接收OutputStream类的实例。这就好比将OutputStream类重新包装了 一下，将一个输出流的实例传到打印流之后，是打印流把输出流重新包装了一下。这种 设计称为装饰者设计模式。

**打印流的格式化输出方法：**

可以使用打印流的printf( )方法进行格式化输出操作，但需要制定其输出的数据类型， 但也可以全部使用%s代替。这种格式化输出类似于C语言的输出语句。

**System类对IO的支持：**

System类对IO的支持指的是System类中的三个常量：out、err、in

1. System.out：out是一个PrintStream类的对象，输出语句中print( )或println( )方法都 是调用的PrintStream类中的print( )或println( )方法。此对象表示的是标准屏幕输出， 用来向显示屏上输出信息。

对于OutputStream类来讲，哪个子类为其实例化，就具备了向哪里输出的能力，如

FileOutputStream类表示向文件输出，System.out表示向显示器输出。

1. System.err：err是一个PrintStream类的对象，表示错误信息输出，如果程序出现错 误，则可以使用System.err.println(e)进行输出。

System.out和System.err都可以输出错误信息，但System.out一般是将信息显示给， 而System.err则正好相反，是不希望用户看到的，会直接在后台打印，是专门显示 错误信息的。所以，如果想要输出错误信息，优先使用System.err输出。

1. System.in：in是一个InputStream类对象，表示键盘输入流，使用System类可以完 成从键盘读取数据的功能。一般与Scanner类和BufferedReader类配合使用。

**输入输出重定向：**

1）可以使用System类中的set( )方法改变System.out和System.err两个输出流的输出 位置以及System.in的输入流来源。

2）例如，修改System.out的输出位置以可以方便把产生的错误信息输出到一个文件中。

3）最好不要err的输出位置。

**BufferedReader类：**

BufferedReader类本质上一种输入流。使用BufferReader类可以从输入流中（增加输 入流、屏幕输入流）读取内容。但BufferedReader中定义的构造方法只能接收字符输入 流的实例，所以必须使用InputStreamReader类将字节输入流转换为字符输入流。

**Scanner类：**

Scanner类是专门的输入数据类。Scanner类不仅可以完成输入操作，而且可以方便地 对输入的数据进行验证。

Scanner类定义在java.util.包中。

**Scanner类的分隔符：**

Scanner默认的分隔符为空格，当读到空格后会结束。所以为了方便，可以使用Scanner 类中的useDelimiter(“\n”)方法自定义Scanner类的分隔符。

**数据操作流：**

数据操作流指的是DataOutputStream类和DataInputStream类。通常情况下，使用数据 输出流（DataOutputStream类）按照一定格式将数据进行输出操作，再通过数据输入流 （DataInputStream类）按照一定的格式将数据读入。

1. DataOutputStream类是OutputStream类的间接子类。
2. DataInputStream类是InputStream类的间接子类。专门负责读取使用 DataOutputStream类输出的数据。

**合并流：**

合并流指的是SequenceInputStream类，它的主要功能是将两个文件的内容合并成一 个文件。

**压缩流：**

使用压缩流可以将文件或文件夹压缩成ZIP、JAR、GZIP等格式的压缩文件。

**ZIP压缩输入/输出简介：**

1）ZIP是一种较为常见的压缩形式，在Java中实现ZIP的压缩需要导入java.util.zip包。

**ZipOutputStream类：**

1）如果要想完成一个文件或文件夹的压缩，可以使用ZipOutputStream类完成，它是 OutputStream类的子类。

2）压缩文件时需要对每个文件设置ZipEntry对象。

3）如果想要对文件夹进行压缩，需要列出文件夹中所有的File类对象，分别设置ZipEntry 对象，完成压缩。

**ZipFile类：**

在Java中，每一个压缩文件都可以使用ZipFile表示，使用ZipFile类可以根据压缩后的 文件名称找到其对应的ZipEntry对象并将其进行解压缩操作。但这种解压缩只适合压缩 文件中只存在一个文件的情况，如果要操作更加复杂的压缩文件，可以使用 ZipInputStream类完成。

**ZipInputStream类：** 1）ZipInputStream类是InputStream类的子类，使用ZipInputStream类可以方便的读取 ZIP格式的压缩文件。

1. 使用ZipInputStream类与ZipFile类结合可以对压缩的文件夹进行解压缩的操作。

**回退输入流：** 回退输入流（PushbackInputStream和PushbackReader类），可以把读取进来的某些数据 重新退回到输入流的缓冲区中。

**字符编码：**

1. 使用System.getProperty(“file.encoding)可以得到本机默认编码；
2. 可以使用String类中的getBytes(String charset)方法实现该字符串的编码转换。

**乱码产生的原因及避免：**

乱码产生的原因是程序指定的编码（即输出内容的编码）与本机默认编码（即接收内容 的编码）不一致。

使用要想避免乱码的产生，只要让程序的编码与本地默认编码一致即可。

**对象的序列化：**

1）对象的序列化是指把一个对象变为二进制的数据流的一种方法，通过对象的序列化 可以方便地实现对象的传输和存储。

2）一个类的对象要想被序列化，则该类必须实现Serializable接口，此接口是一个标识 接口，表示该类对象具备了被序列化的能力。

3）完成对象的输入输出还需要依靠对象输出流(ObjectOutputInputStream类)和对象输入 流(ObjectInputStream类)。

4）使用对象输出流输出对象的过程有时也称对象的序列化，而是由对象输入流读取对 象的过程有时也称反序列化。

5）对象序列化只有对象的属性被序列化了。

**ObjectOutputStream类：** 1）如果一个对象要想被输出，则必须使用ObjectOutputStream类。它是OutputStream 类的子类。

2）在实例化对象输出流时需要传入OutputStream类子类对象，根据传入不同的 OutputStream子类对象，其输出的位置也不同。

**ObjectInputStream类：**

1. 使用ObjectInputStream类可以直接把序列号的对象反序列化。它是InputStream类的子类。
2. 在实例化ObjectInputStream类对象时，需要传入InputStream类子类对象，根据传入不同的InputStream子类对象，其输入的来源也不同。

**transient关键字：**

当使用Serializable接口实现序列化时，如果一个对象中的某个属性不希望被序列 化的话，则可以使用transient关键字进行声明。

序列化一组对象：

如果需要序列化多个对象，就可以使用对象数组进行操作，因为数组属于引用数据类型， 所以可以直接使用Object类型进行接收。