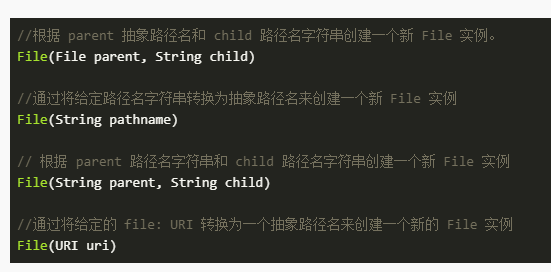
1. Java 使用 File 类来直接处理文件和文件系统。File 类没有指定信息怎样从文件读取或向文件存储；它描述了文件本身的属性。File 对象用来获取或处理与磁盘文件相关的信息，例如权限，时间，日期和目录路径。此外，File 还浏览子目录层次结构。Java 中的目录当成 File 对待，它具有附加的属性——一个可以被 list( )方法检测的文件名列表。
   1. File 类提供了以下构造方法：





1. java.io 包中提供了文件操作类：

用于读写本地文件系统中的文件：FileInputStream 和 FileOutputStream

描述本地文件系统中的文件或目录：File、FileDescriptor 和 FilenameFilter

提供对本地文件系统中文件的随机访问支持：RandomAccessFile

1. FileInputStream 类用于打开一个输入文件，若要打开的文件不存在，则会产生异常 FileNotFoundException，这是一个非运行时异常，必须捕获或声明抛弃；

FileOutputStream 类用来打开一个输出文件，若要打开的文件不存在，则会创建一个新的文件，否则原文件的内容会被新写入的内容所覆盖；

在进行文件的读/写操作时，会产生非运行时异常 IOException，必须捕获或声明抛弃（其他的输入/输出流处理时也同样需要进行输入/输出异常处理）。

1. 如果文件内容保存的是字符信息，如 txt 文件等，还可以使用 FileReader 来读取文件内容。
2. 对于 FileInputStream/FileOutputStream、FileReader/FileWriter 来说，它们的实例都是顺序访问流，即只能进行顺序读/写。而类 RandomAccessFile 则允许文件内容同时完成读和写操作，它直接继承 object，并且同时实现了接口 DataInput 和 DataOutput。
3. RandomAccessFile 提供了支持随机文件操作的方法：
   1. readXXX()或者 writeXXX():如 ReadInt(),ReadLine(),WriteChar(),WriteDouble()等
   2. int skipBytes(int n):将指针向下移动若干字节
   3. length():返回文件长度
   4. long getFilePointer():返回指针当前位置
   5. void seek(long pos):将指针调用所需位置
4. 可以使用 Files 工具类的 copy(Path source,Path target,CopyOption... options)拷贝文件或者目录。如果目标文件存在，那么赋值将失败，除非我们在 options 中指定了REPLACE\_EXISTING属性。当该命令复制目录时，如果目录中已经有了文件，目录中的文件将不会被复制。CopyOption 参数支持以下 StandardCopyOption 和 LinkOption 枚举：

REPLACE\_EXISTING - 即使目标文件已存在，也执行复制。如果目标是符号链接，则复制链接本身（而不是链接的目标）。如果目标是非空目录，则复制将失败并显示 FileAlreadyExistsException 异常。

COPY\_ATTRIBUTES - 将与文件关联的文件属性复制到目标文件。支持的确切- 文件属性是文件系统和平台相关的，但 last-modified-time 跨平台支持并复制到目标文件。 NOFOLLOW\_LINKS - 表示不应遵循符号链接。如果要复制的文件是符号链接，则复制链接（而不是链接的目标）。

1. Files 类的 move(Path, Path, CopyOption... options)方法移动文件或者目录，同样目标目录存在，那么比如使用REPLACE\_EXISTING。 options 参数支持 StandardCopyOption 的以下枚举：

REPLACE\_EXISTING - 即使目标文件已存在，也执行移动。如果目标是符号链接，则替换符号链接，但它指向的内容不受影响。

ATOMIC\_MOVE - 将移动作为原子文件操作执行。如果文件系统不支持原子移动，则抛出异常。使用，ATOMIC\_MOVE 您可以将文件移动到目录中，并保证观察目录的任何进程都可以访问完整的文件。

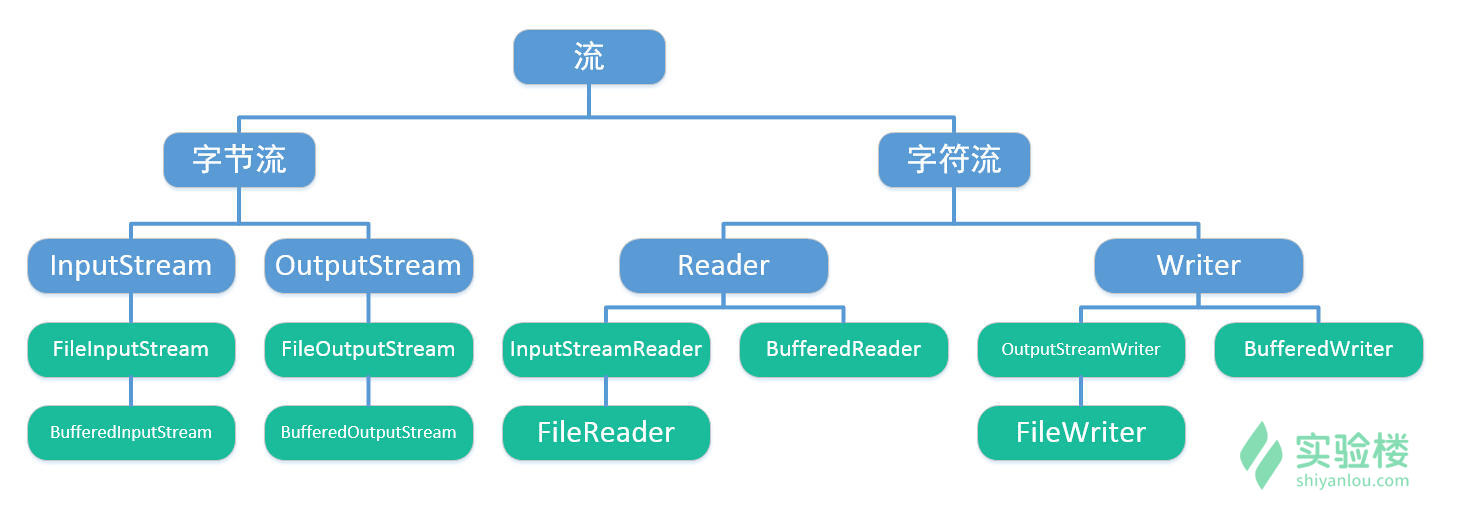
move 方法除了可以移动之外，也可以用与重命名。

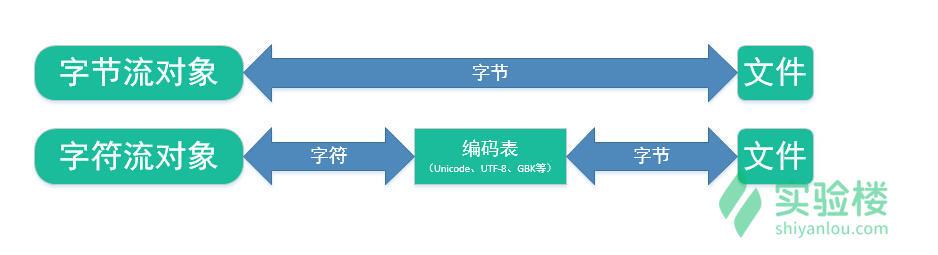
1. 可以通过 Files 的 delete(Path path)方法或者 deleteIfExists(Path path)方法删除文件。
2. Java 使用 File 类表示文件或者目录，可以通过 File 类获取文件或者目录的相关属性。
3. 在大多数程序中，都需要对输入输出进行处理。例如我们中需要获取用户从键盘上的输入，需要在控制台输出结果等等。除此之外还有从文件中读取数据，向文件中写入数据等等。在 Java 中，我们把这些不同类型的输入输出源抽象地称为流，也就是Stream；在里面输入输出的数据则称为数据流（Data Stream），它们通常具有统一的接口。
4. 于是我们得到了数据流的定义：

一个 Java I/O 对象叫做数据流。读取数据到内存的对象叫做输入流，内存写出数据的对象叫做输出流。

1. 针对其面向的不同角度，我们大致可以将流分为下面几种类型：
   1. 按照数据流的方向不同分为输入流和输出流。这种分类不是绝对的，例如在向一个文件写入数据时，它就是输出流；而在读取数据时，它就是输入流。
   2. 按照处理数据的单位不同分为字节流和字符流。
   3. 按照功能的不同分为节点流和处理流。
   4. 节点流是从特定的数据节点（文件、数据库、内存等）读写数据；处理流是连接在已有的流上，通过对数据的处理为程序提供更多功能。
2. 字节流：表示以字节为单位从 stream 中读取或往 stream 中写入信息。通常用来读取二进制数据。

字符流：以 Unicode 字符为单位从 stream 中读取或往stream 中写入信息。





1. 字节流主要操作 byte 类型数据，以 byte 数组为准，java 中每一种字节流的基本功能依赖于基本类 InputStream 和 Outputstream，他们是抽象类，不能直接使用。字节流能处理所有类型的数据（如图片、avi等）。
2. InputStream 是所有表示字节输入流的父类，继承它的子类要重新定义其中所定义的抽象方法。InputStream 是从装置来源地读取数据的抽象表示，例如 System 中的标准输入流 in 对象就是一个 InputStream 类型的实例。



1. 在 InputStream 类中，方法 read() 提供了三种从流中读数据的方法：
   1. int read()：从输入流中读一个字节，形成一个 0~255 之间的整数返回（是一个抽象方法）
   2. int read(byte b[])：从输入流中读取一定数量的字节，并将其存储在缓冲区数组 b 中。
   3. int read(byte b[],int off,int len)：从输入流中读取长度为 len 的数据，写入数组 b 中从索引 off 开始的位置，并返回读取得字节数。
   4. 对于这三个方法，若返回-1，表明流结束，否则，返回实际读取的字符数。
2. OutputStream 是所有表示位输出流的类之父类。子类要重新定义其中所定义的抽象方法，OutputStream 是用于将数据写入目的地的抽象表示。例如 System 中的标准输出流对象 out 其类型是java.io.PrintStream，这个类是 OutputStream 的子类。



1. 字符流以字符为单位，根据码表映射字符，一次可能读多个字节，只能处理字符类型的数据。

java.io 包中专门用于字符流处理的类，是以 Reader 和 Writer 为基础派生的一系列类。

同类 InputStream 和 OutputStream 一样，Reader 和 Writer 也是抽象类，只提供了一系列用于字符流处理的接口。它们的方法与类 InputStream 和 OutputStream 类似，只不过其中的参数换成字符或字符数组。

Reader 是所有的输入字符流的父类，它是一个抽象类。





1. 对字符数组进行处理： CharArrayReader、CharArrayWrite

对文本文件进行处理：FileReader、FileWriter

对字符串进行处理：StringReader、StringWriter

过滤字符流：FilterReader、FileterWriter

管道字符流：PipedReader、PipedWriter

行处理字符流：LineNumberReader

打印字符流：PrintWriter

1. InputStreamReader 和 OutputStreamWriter 是 java.io 包中用于处理字符流的最基本的类，用来在字节流和字符流之间作为中介：从字节输入流读入字节，并按编码规范转换为字符；往字节输出流写字符时先将字符按编码规范转换为字节。使用这两者进行字符处理时，在构造方法中应指定一定的平台规范，以便把以字节方式表示的流转换为特定平台上的字符表示。
2. 类 BufferedInputStream 和 BufferedOutputStream 实现了带缓冲的过滤流，它提供了缓冲机制，把任意的 I/O 流“捆绑”到缓冲流上，可以提高 I/O 流的读取效率。

在初始化时，除了要指定所连接的 I/O 流之外，还可以指定缓冲区的大小。缺省时是用32字节大小的缓冲区；最优的缓冲区大小常依赖于主机操作系统、可使用的内存空间以及机器的配置等；一般缓冲区的大小为内存页或磁盘块等的整数倍。

BufferedInputStream 的数据成员 buf 是一个位数组，默认为 2048 字节。当读取数据来源时例如文件，BufferedInputStream 会尽量将 buf 填满。当使用 read ()方法时，实际上是先读取 buf 中的数据，而不是直接对数据来源作读取。当 buf 中的数据不足时，BufferedInputStream 才会再实现给定的 InputStream 对象的 read() 方法，从指定的装置中提取数据。

BufferedOutputStream 的数据成员 buf 是一个位数组，默认为 512 字节。当使用 write() 方法写入数据时，实际上会先将数据写至 buf 中，当 buf 已满时才会实现给定的 OutputStream 对象的 write() 方法，将 buf 数据写至目的地，而不是每次都对目的地作写入的动作。

1. 同样的，为了提高字符流处理的效率，java.io 中也提供了缓冲流 BufferedReader 和 BufferedWrite。其构造方法与 BufferedInputStream 和 BufferedOutPutStream 相类似。另外，除了 read() 和 write() 方法外，它还提供了整行字符处理方法：

public String readLine()：BufferedReader 的方法，从输入流中读取一行字符，行结束标志\n、\r或者两者一起（这是根据系统而定的）

public void newLine()：BufferedWriter 的方法，向输出流中写入一个行结束标志，它不是简单地换行符\n或\r，而是系统定义的行隔离标志（line separator）

1. 数据流类 DataInputStream 和 DataOutputStream 的处理对象除了是字节或字节数组外，还可以实现对文件的不同数据类型的读写：
   1. 分别实现了 DataInput 和 DataOutput 接口
   2. 在提供字节流的读写手段同时，以统一的形式向输入流中写入 boolean，int，long，double 等基本数据类型，并可以再次把基本数据类型的值读取回来。
   3. 提供了字符串读写的手段
2. 可以通过 ObjectOutputStream 和 ObjectInputStream 将对象输入输出。 将对象的状态信息转换为可以存储或者传输的形式的过程又叫序列化。
3. Java NIO(New IO)发布于 JDK1.4，用于代替 Java 标准 IO 。Java NIO是面向缓存的、非阻塞的IO，而标准IO是面向流的，阻塞的IO。

首先理解 NIO 的重要概念-Buffer（缓冲区）

NIO 读取或者写入数据都要通过 Buffer

通过 allocate()方法分配 Buffer，Buffer 不可实例化，Buffer 是抽象类，需要使用具体的子类，比如 ByteBuffer。

Buffer 的参数

- capacity ：缓冲区的容量

- position ：当前指针位置，没读取一次缓冲区数据或者写入缓冲区一个数据那么指针将会后移一位

- limit ：限制指针的移动，指针不能读取 limit 之后的位置 ​ - mark ：如果设置该值，那么指针将移动到 0~position 的位置

- 最后可以这几个参数的关系如下：mark <= position <= limit <= capacity