File类

1. 用于新建、删除、重命名文件和目录，**但不可以访问文件本身**；
2. 常用方法：

> - 使用文件路径字符串创建File实例；
> - 访问文件名：getName()返回文件名；getPath()返回路径名；renameTo()重命名；
> - 文件检测：exists()是否存在；canWrite()是否可写；canRead()是否可读；isFile()是否是文件而不是目录；
> - 获取常规文件信息：**length()获取文件内容长度**；
> - 文件操作：createNewFile()若不存在file对象，则创建新文件；delete()删除该对象文件；deleteOnExit()在程序退出时删除该文件；
> - 目录操作：list()列出所有子文件名和路径名，可以选择返回值为String数组和File数组

文件过滤器 list方法可以接受一个FilenameFilter参数，函数式接口，包含一个boolean accept(File dir, String name);方法，可以用来选择返回的文件； 在这个方法中，把本file对象作为参数dir传入，把**file对象中的所有子文件名和路径名作为参数name**一个个传入；

流

1. 输入输出的概念是针对程序来说的，把数据写入程序中的叫输入流；
2. 分为字符流和字节流；字符流：Reader和writer；字节流：InputStream和OutputStream；
3. 节点流和处理流：节点流就是直接以数据源做构造器参数；而处理流是用流做构造器参数；
4. 流等**IO资源不属于内存里的资源**，需要显示关闭；

输入流通用方法：read()

1. 字符流Reader中，read()取出单个字符，**返回int类型**；可输入参数为char[];
2. 字节流InputStream中，read()取出单个字节，**返回int类型**，可输入参数为byte[];
3. read(数组，int off,int len),指**读取len个数据，从数组off位置开始**放入数组中；
4. **当使用数组做参数读取多个字符或字节时，返回的是读取的数据数**； 还支持：

- void mark(int readAheadLimit)在当前指针记录一个标记(mark);
- void reset():把指针定位到上一次记录标记(mark)的位置；
- long skip(long n)：使记录指针向前移动n个数据；

输出流通用方法：write()

1. 字节流使用int作为参数；字符流使用String或者int作为参数；
2. 字节流使用btye[]数组作为参数； 字符流用char[]数组做参数；
3. 关闭数据流能让缓冲区数据flush到节点中。

转换流：**只有将字节流转换为字符流**，因为字符流方便

1. InputStreamReader和OutputStreamWriter；

BufferedReader有一个readline()方法可以方便的一次读入一行内容，所以把使用来读取文本内容的输入流包装成它；

推回输入流：PushbackInputStream和PushbackReader

1. 有一个推回缓冲区。会先从推回缓冲区中读取，只有完全读取后才从数据流中读；

标准输入、输出 System.in是InputStream； System.out是PrintStream；

虚拟机读写其他进程的数据

1. Runtime.getRuntime().exec()可以运行平台上的别的程序，产生一个Process对象；
2. Process对象提供三个方法：

- getOutputStream();获取子进程输出流；输入输出还是对当前执行进程说的，所以该流可以向别的进程写输入，即别的进程从当前进程读数据；
- getInputStream();获取子进程输入流；
- getErrorStream();获取子进程错误留。

RandomAccessFile类

1. 只需要访问文件部分内容而不是从头到尾读，使用RandomAccessFile更好；
2. 向已存在的文件后追加内容，也是更好，因为RandomAccessFile允许自有定位文件记录指针（mark）；
3. 只能读文件；
4. 提供两个方法操作读取指针：

- long getFilePointer()：返回指针位置；
- void seek(long pos)：指针定位到pos位置；

5. 构造器：两个参数：（1）String文件名或者file对象；（2）mode参数；

- mode参数是指定访问模式："r"只读；"rw"读写,如果文件不存在会创建该文件；"rws"？"rwd"?

6. 通过length()方法获取内容长度后可以设置指针位置来把内容添加到文件原内容后；
7. 如果指针在内容中间，新输出内容会覆盖文件原有内容！；

多断点网络下载工具实现；

对象序列化

1. 序列化目标：将对象保存在磁盘中；手段:把java对象转换成平台无关的二进制流，从而允许这种二进制流持久地保存在磁盘上。
2. 对象的序列化具体实现是把java对象写入IO流中；反序列化是从IO流中恢复java对象；
3. 要使对象可序列化，必须实现**Serializable**和**Externalizable**两个接口，只是标记接口，无须实现任何方法；
4. 注意：一个可序列化类有多个父类时，其父类要满足：（**1**）有无参数的构造器或者（2）可序列化；否则会抛异常；如果只满足第一个条件，则父类定义的成员变量值不会被序列化（即不会出现在二进制流中）；

序列化某对象的步骤：

1. 创建一个ObjectOutputStream流，这个流是处理流，即一定要有写入的目标文件；
2. 调用ObjectOutputStream对象的**writeObject(Object obj)**方法输出对象。

反序列化对象的步骤：

1. 创建一个ObjectInputStream流，这个流是处理流;

2. 调用ObjectOutputStream对象的**readObject()**方法读取流中的对象,该方法返回一个Object对象（读取后要类型转换）。 反序列化机制无须通过构造器来初始化对象。

当有引用类型X的变量在序列化对象M中时

1. 该引用类型X必须也要可序列化，否则**M对应的类不可以被序列化！**；
2. 当多个对象中有一变量指向同一个引用对象时（即有对象a、b、c，该类有一个引用类型变量len，a.len==b.len==c.len），会采用特殊算法

> - 每个被序列化的对象都会有一个编号；
> - 序列化该对象时会先检查，只有没有被序列化过才会被序列化；
> - 如果被序列化过，则**只是输出一个编号**，而不是输出一个对象的二进制。

3. 从2中会延伸出一个问题，比如：序列化顺序为abc，序列化a后，修改了b.len中的某一个变量，则序列化b时，该变量会不会把改变带入到序列列中？

> 答案是：不会，因为b.len实际被序列化过，后面的只会输出一个编号。

自定义序列化

1. 使用transient关键字修饰实例变量（**只能**），则该实例变量不会被序列化；
2. 自定义序列化机制：**需要序列化的类**继承了Serializable接口后，重写三个方法可自定义如何序列化

> - private void writeObject(ObjectOutputStream out) //序列化时会调用此方法
> - private void readObject(ObjectInputStream in) //反序列化时会调用此方法
> - private void readObjectNoData() //当序列化的类版本和反序列化的类版本不同时，或者ObjectInputStream流被修改时，会调用此方法。 writeObject方法中序列化变量的顺序和readObject恢复的顺序要一致

序列化的版本问题！ 使用 private static final long serialVersionUID = XX;用于表示序列化版本。 如果不显示指定的话，jvm会自动计算，但不同jvm之间计算方法不同，所以容易反序列化失败。

NIO

1. 面向流的输入输出系统，如果没有读到有效的数据会阻塞线程的执行；
2. NIO使用**内存映射文件**来处理输入输出，将文件或者文件的一段区域映射到内存中，然后可以像访问内存一样访问文件。
3. Channel（通道）和Buffer（缓冲）是NIO的核心对象；
4. 所有数据都要通过通道Channel传输，与流区别在于使用map()方法，将一块数据映射到内存中，是**面向块**的处理；
5. Buffer本质是一个数组，发送到Channel的对象都要先放到Buffer中；读之前也要先放Buffer中。

Buffer

1. Buffer从内部结构上是一个数组，可以保存多个类型相同的数据，有多种Buffer类型对应每种基础类型都有一种(boolean除外)，常用的是：ByteBuffer和CharBuffer；
2. **使用静态方法**： XXXBuffer.allocate(int capacity)创建容量为capacity的buffer对象；
3. Buffer中有三个概念：容量(capacity)，界限（limit）和位置（position）；

> - 界限是指limit后的数据不可读写；
> - 位置（position）是读取指针位置。

4. Buffer主要作用是转入数据然后输出数据。
5. 使用：

> - **装入数据：**使用put(obj)方法向Buffer放入数据，position会向后移；
> - **获取数据：**调用flip()方法，会将limit设置到positioin的位置，所以之后Buffer就不可以写入新数据；
> - **读取数据结束后：调用clear()方法，该方法不会清除数据，只会把position设为0，limit设置为容量capacity**;

6. 读取数据使用**get()方法**，可以使用一个int参数，来获取指定位置的数据；
7. 有一种叫：直接Buffer，创建成本高，但读取效率高，适合长生存期？

Channel

1. 作用：将文件部分或全部映射成Buffer
2. **不通过构造器来创建**，通过流节点（OutputStream这些）的getChannel()方法返回对应的Channel；
3. 三种方法：map()、read()和write();后两种方法的参数都是Buffer类型；
4. 读取文件：先用read(buffer),然后使用buffer.get()获取数据；类似流中的read(数组)；写入文件也类似
5. map(MapMode mode,long position, long size)方法返回一个buffer类型，其参数mode是执行映射时的模式，分为：只读、读写等；第二第三参数控制多少数据放入buffer中与在buffer中的位置；（mode一般会有类变量参数）

字符集和Charset

1. 字符集就是Unicode这些；
2. 静态方法Charset.availableCharsets()能获取JDK支持的所有字符集；
3. 使用System.getProperty("file.encoding")可以获取系统所用的编码字符集；
4. Charset是**通过Charset.forName(字符集)创建对象的**；
5. 解码器和编码器：CharsetDecoder和CharsetEncoder;通过Charset对象的newDecoder()和newEncoder()生成。
6. 解码器和编码器分别有decode()和encode()两个方法，把ByteBuffer（字节序列）转化为CharBuffer（字符序列）或者反过来；
7. StandardCharsets类有**类变量**来代表各种字符集的Charset对象；
8. 如果只需要简单的转码，则可以直接调用Charset的decode()和encode()方法；

文件锁

Paths和Files工具类