Java Day 18

线程扩展内容

可重入锁

---

java.util.concurrent.locks.ReentrantLock-->ReentrantLockTest.java

ReentrantLock 类实现了 Lock，它拥有与 synchronized 相同的并发性和内存语义，但是添加了类似锁投票、定时锁等候和可中断锁等候的一些特性。它还提供了在激烈争情况下更佳的性能。

1）同步

使用 lock()和 unlock()方法进行同步，unlock()操作放到 fianlly 中

2）通信

使用 ReentrantLock 类的 newCondition()方法可以获取 Condition 对象

需要等待的时候使用 Condition 的 await()方法，唤醒的时候用 signal()方法

不同的线程使用不同的Condition，这样就能区分唤醒的时候找哪个线程了

读写锁

ReentrantReadWriteLock，读写锁的机制：-->ReadWriteLockTest.java

"读-读"不互斥，提高效率

"读-写"互斥

"写-写"互斥

线程组-->ThreadGroupTest.java

Java 中使用 ThreadGroup 来表示线程组，它可以对一批线程进行分类管理，Java 允许程序直接对线程组进行控制

默认情况下，所有的线程都属于主线程组

线程池-->ThreadPoolTest.java

程序启动一个新线程成本是比较高的，因为它涉及到要与操作系统进

行交互。而使用线程池可以很好的提高性能，尤其是当程序中要创建大量生存期很短的线程时，更应该考虑使用线程池。线程池里的每一个线程代码结束后，并不会死亡，而是再次回到线程池中成为空闲状态，等待下一个对象来使用。从 JDK5 开始，Java 内置支持线程池

JDK5 新增了一个 Executors 工厂类来产生线程池：

1）newCachedThreadPool 创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。

2）newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。

3）newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。

4）newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO，LIFO，优先级)执行。

这些方法的返回值是 ExecutorService 对象，该对象表示一个线程池，可以执行 Runnable 对象或者 Callable 对象代表的线程任务：

```
Future<?> submit(Runnable task)
<T> Future<T> submit(Callable<T> task)
```

使用步骤：

1）创建线程池对象

2）创建 Runnable/Callable 实例

3）提交 Runnable/Callable 实例

4）关闭线程池

Callable 方式向线程池添加任务-->ThreadPoolTest2.java

定时任务调度：

Unix/Linux 中的 crontab

java.util.Timer-->TimerTest.java
java.util.concurrent.ScheduledExecutorService-->TimerTest2.java

使用开源类库 Quartz、JCronTab

1）Timer 的优点在于简单易用，但由于所有任务都是由同一个线程来调度，因此所有任务都是串行执行的，同一时间只能有一个任务在执行，前一个任务的延迟或异常都将会影响到之后的任务。如果说某

个任务出现了异常，这个定时器上的所有任务都会终止。

2）ScheduledExecutor 每一个被调度的任务都会由线程池中一个线程去执行，因此任务是并发执行的，相互之间不会受到干扰。

volatile：修饰的变量是易失（不稳定）变量

1）线程可以把成员变量保存在线程工作内存（缓存）中，而不是直接在主存中进行读写。这就可能造成一个线程在主存中修改了一个变量的值，而另外一个线程还继续使用线程工作内存中的变量值的拷贝，造成数据的不一致。

volatile 修饰的成员变量在每次被线程访问时，都强迫从主内存中重读该成员变量的值。当成员变量发生变化时，强迫线程将变化值回写到主内存。

2）当我们把代码写好之后，虚拟机不一定会按照我们写的代码的顺序来执行。虚拟机在进行代码编译优化的时候，对于那些改变顺序之后不会对最终变量的值造成影响的代码，是有可能将他们进行指令重排序的。

如果一个变量被声明 volatile 的话，那么这个变量不会被进行指

令重排序。

3）原子操作：即一个操作或者多个操作，要么全部执行并且执行的过程不会被任何因素打断，要么就都不执行。由于 Java 中的运算并非是原子操作，所以导致 volatile 声明的变量无法保证线程安全。

volatile 对于一写多读，是可以解决变量同步问题，但是如果多写，同样无法解决线程安全问题。

总结：volatile 具有可见性、有序性，不具备原子性。

何时使用 volatile？java 中多线程共享的数据，并且不是放到 synchronized 代码块中进行操作，一写多读

volatile 不会让线程阻塞，响应速度比 synchronized 高

查看汇编指令：javap -c classes/com/briup/chap10/Account

IO:Input、Output

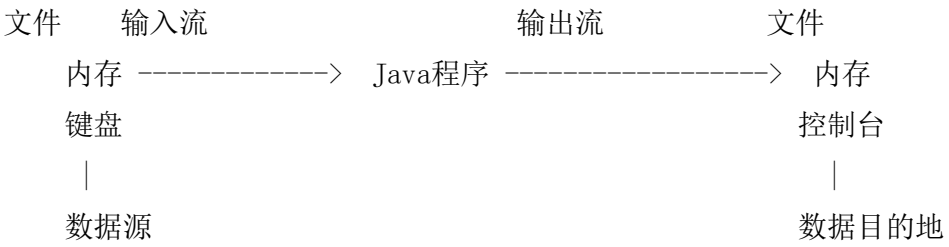
流：有序的数据序列

IO 流用来处理设备之间的数据传输

Java 对数据的操作是通过流的方式

Java 用于操作流的类都在 java.io 包中

流按流向分为两种：输入流，输出流。



流按操作最小数据单元类型分为两种：

字节流 ：字节流可以操作任何数据，因为在计算机中任何数据都是以字节的形式存储的

字符流 ：字符流只能操作纯字符数据，比较方便

IO 流常用父类

字节流的抽象父类：

InputStream
OutputStream

字符流的抽象父类：

Reader
Writer

FileInputStream,FileOutputStream 使用

---

read()一次读取一个字节，返回值是 int，不是 byte，只用低 8 位

因为字节输入流可以操作任意类型的文件,比如图片音频等,这些文件底层都是以二进制形式的存储的,如果每次读取都返回 byte,有可能在读到中间的时候遇到111111111,那么这11111111是byte类型的-1,我们的程序是遇到-1 就会停止不读了,后面的数据就读不到了,所以在读取的时候用 int 类型接收,如果 11111111 会在其前面补上 24 个 0 凑足 4 个字节,那么 byte 类型的-1 就变成 int 类型的 255 了，这样可以保证整个数据读完,而结束标记的-1 就是 int 类型

write(int)一次写出一个字节，虽然写的是 int，但到文件中的是 byte，会自动去除前 24 位

FileOutputStream 如果文件不存在，新建空文件；如果文件已经存在，清空其内容；如果想向文件追加内容，将构造器第二个参数传 true

FileInputStream.available()：返回可读字节数

read(byte[])：一次读多个字节

wrire(byte[])：一次写多个字节

write(byte[] b, int off, int len)写出有效的字节个数

字节流一次读写一个数组的速度明显比一次读写一个字节的速度快很多，这是加入了数组缓冲区的效果

Filter 相关的流，是辅助流，需要依赖其它流，起到功能增强的作用

BufferedInputStream-->Copy2.java

BufferedInputStream 内置了一个缓冲区(数组)

从 BufferedInputStream 中读取一个字节时

BufferedInputStream 会一次性从文件中读取 8192 个字节，存在缓冲区中，返回给程序一个；程序再次读取时，就不用找文件了，直接从缓冲区中获取，直到缓冲区中所有的都被使用过，才重新从文件中读取 8192 个字节

BufferedOutputStream

BufferedOutputStream 也内置了一个缓冲区(数组)

程序向流中写出字节时，不会直接写到文件，先写到缓冲区中，直到缓冲区写满，BufferedOutputStream 才会把缓冲区中的数

据一次性写到文件里

flush()方法：用来刷新缓冲区的,刷新后可以再次写出

close()方法：用来关闭流释放资源的,如果是带缓冲区的流对象的close()方法,不但会关闭流,还会在关闭流之前刷新缓冲区,关闭后不能再写出