链滴

lock 和 synchronized

作者：JackMeiiii

原文链接：https://ld246.com/article/1586426364316

来源网站：链滴

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20180119.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
一、为什么分布式环境下 synchronized 失效？ 因为不同服务属于不同进程
synchronized：底层是通过 moniterrenter\moniterexit 指令来完成，JVM 需要保证每一个 m
nitorenter 都有一个 monitorexit 与之相对应
使用：

修饰实例方法，作用于当前实例加锁，进入同步代码前要获得当前实例的锁。</stron
>
静态方法，作用于当前类对象加锁，进入同步代码前要获得当前类对象的锁。</stron
>
修饰代码块，指定加锁对象，对给定对象加锁，进入同步代码库前要获得给定对象的
。

<img src="https://ld246.com/images/img-loading.svg" alt="image.png" data-src="https:
/b3logfile.com/file/2020/04/image-d5196786.png?imageView2/2/interlace/1/format/jpg"></
>
作用：解决并发编程中存在的原子性、可见性和有序性问题

原子性：一个操作是不可中断的，要么全部执行成功要么全部执行失败</li

可见性：当一个线程修改了共享变量后，其他线程能够立即得知这个修改（synchroniz
d、volatile 修改值后都会将共享变量同步到主内存中）
有序性：指执行代码是有序的去执行，线程 1 先执行，完了之后线程 2 才去执行（因为
JAVA 中允许重排序只要不改变执行结果，数据之间不存在依赖）比如：instance = new Singleton();
1.分配对象的内存空间；2.初始化对象；3.设置 instance 指向刚分配的内存地址，如果 2、3 重排，
么并发时候判断 instance 是否== null,就会出现错误判断 ~~（synchronized 只是保证了代
执行的有序性，并没有保证不能编译器指令不会重排）~~

缺点：

不可中断
性能较差(看看后面是怎么改进的)

特性：

可重入：指的是同一线程的外层方法获得获得锁之后,内层方法可以直接在此获取该锁
不可中断：一旦这个锁已经被别人获得了,如果我还想获取,我只能选择等待或者阻塞,直到别的线
释放这个锁(返回或者抛异常).如果别人永远不释放锁,那么我只能永远地等下去<
r>
相比之下,Lock 类,可以拥有中断的能力,
第一点:如果我觉得我等的时间太长了,有权中断现在已经获取到锁的线程的执行,
第二点:如果我绝得我等待的时间太长不想再等了,也可以退出.
不需要手动释放

二、为什么有了 synchronized 还需要 volatile

一方面是因为 synchronized 是一种锁机制，存在阻塞问题和性能问题，而 volatile 并不是锁，
以不存在阻塞和性能问题
volatile 禁止重排的功能，不可替代 （synchronized 只是保证了代码执行的有序性，并没有保
不能编译器指令不会重排）

三、为什么 volatile 不能保证原子性

因为定义一个 volatile 变量后，如果此变量在多线程情况下执行类似像 i++ 这种不是原子操作，导致最终值不对。~~比如~~~~当 i=5 的时候 A,B 两个线程同时读入了 i 的值， 然后 A 线程执行了 emp = i + 1 的操作， 要注意，此时的 i 的值还没有变化，然后 B 线程也执行了 temp = i + 1 的操，注意，此时 A，B 两个线程保存的 i 的值都是 5，temp 的值都是 6， 然后 A 线程执行了 i = temp（6）的操作，此时 i 的值会立即刷新到主存并通知其他线程保存的 i 值失效， 此时 B 线程需要重新取 i 的值那么此时 B 线程保存的 i 就是 6，同时 B 线程保存的 temp 还仍然是 6， 然后 B 线程执行 i= emp （6），所以导致了计算结果比预期少了 1。~~

四、volatile 和 synchronized 的区别

volatile 本质是在告诉 jvm 当前变量在寄存器（工作内存）中的值是不确定的，需要从主存中读； synchronized 则是锁定当前变量，只有当前线程可以访问该变量，其他线程被阻塞住。
volatile 仅能使用在变量级别；synchronized 则可以使用在变量、方法、和类级别的
volatile 仅能实现变量的修改可见性，不能保证原子性；而 synchronized 则可以保证变量的修可见性和原子性
volatile 不会造成线程的阻塞；synchronized 可能会造成线程的阻塞。
volatile 标记的变量不会被编译器优化；synchronized 标记的变量可以被编译器优化
