Java 内存模型与线程

Java 内存模型的主要目标是定义程序中各个变量的访问规则，即在虚拟机中将变量存储到内存和从内存中取出变量这样的底层细节。

渤海银行 IC 卡--预授权完成撤销报 96，在联迪 A8 上也是这样的。（系统异常）
农业银行 IC 卡--预授权完成撤销报 40，在联迪 A8 上也是这样的。（请联系发卡行）
上海银行 IC 卡--预授权报 03，商户未登记，换一个终端就好了，给出的解释是上送给银联的 mcc，该商户不支持预授权功能。
邮政储蓄 IC 卡--预授权完成撤销报 05，身份认证失败，换一个终端就好了，至于原因盛付通正在查，有结果会告诉我们。
关于扫码类交易，先退货，后撤销是可以成功的，原因是盛付通后台的问题，他们已经知道这个 bug 了。

1. 内存间交互操作？

lock（锁定）：作用于主内存的变量，它把一个变量标识为一条线程独占的状态。
unlock（解锁）：作用于主内存的变量，它把一个处于锁定状态的变量释放出来，释放后的变量才可以被其他线程锁定。
read（读取）：作用于主内存的变量，它把一个变量的值从主内存传输到线程的工作内存中，以便随后的 load 动作使用。
load（载入）：作用于工作内存的变量，它把 read 操作从主内存中得到的变量值放入工作内存的变量副本中。
use（使用）：作用于工作内存的变量，它把工作内存中一个变量的值传递给执行引擎，每当虚拟机遇到一个需要使用到变量的值的字节码指令时将会执行这个操作。
assign（赋值）：作用于工作内存的变量，它把一个从执行引擎接收到的值赋给工作内存的变量，每当虚拟机遇到一个给变量赋值的字节码指令时执行这个操作。
store（存储）：作用于工作内存的变量，它把工作内存中一个变量的值传送到主内存中，以便随后的 write 操作使用。
write（写入）：作用于主内存的变量，它把 store 操作从工作内存中得到的变量的值放入主内存的变量中。
如果要把一个变量从主内存复制到工作内存，那就要顺序地执行 read 和 load 操作，如果要把变量从工作内存同步回主内存，就要顺序地执行 store 和 write 操作。注意，Java 内存模型只要求上述两个操作必须按顺序执行，而没有保证是连续执行。也就是说，read 与 load 之间、store 与 write 之间是可插入其他指令的，如对主内存中的变量 a、b 进行访问时，一种可能出现顺序是 read a、read b、load b、load a。

在 Java 中，Thread.yield（）可以让出执行时间，但是要获取执行时间的话，线程本身是没有什么办法的。Java 使用的线程调度方式就是抢占式调度。

2. 线程的状态转换？

Java 语言定义了 5 种线程状态，在任意一个时间点，一个线程只能有且只有其中的一种状态，这 5 种状态分别如下。

新建（New）：创建后尚未启动的线程处于这种状态。
运行（Runable）：Runable 包括了操作系统线程状态中的 Running 和 Ready，也就是处于此状态的线程有可能正在执行，也有可能正在等待着 CPU 为它分配执行时间。
无限期等待（Waiting）：处于这种状态的线程不会被分配 CPU 执行时间，它们要等待被其他线程显式地唤醒。以下方法会让线程陷入无限期的等待状态：

- 没有设置 Timeout 参数的 Object.wait（）方法。
- 没有设置 Timeout 参数的 Thread.join（）方法。
- LockSupport.park（）方法。

限期等待（Timed Waiting）：处于这种状态的线程也不会被分配 CPU 执行时间，不过无须等待被其他线程显式地唤醒，在一定时间之后它们会由系统自动唤醒。以下方法会让线程进入限期等待状态：

- Thread.sleep（）方法。
- 设置了 Timeout 参数的 Object.wait（）方法。
- 设置了 Timeout 参数的 Thread.join（）方法。
- LockSupport.parkNanos（）方法。
- LockSupport.parkUntil（）方法。

阻塞（Blocked）：线程被阻塞了，“阻塞状态”与“等待状态”的区别是：“阻塞状态”在等待着获取到一个排他锁，这个事件将在另外一个线程放弃这个锁的时候发生；而“等待状态”则是在等待一段时间，或者唤醒动作的发生。在程序等待进入同步区域的时候，线程将进入这种状态。
结束（Terminated）：已终止线程的线程状态，线程已经结束执行。

线程安全与锁优化

1. 线程安全的概念。

当多个线程访问一个对象时，如果不用考虑这些线程在运行时环境下的调度和交替执行，也不需要进行额外的同步，或者在调用方进行任何其他的协调操作，调用这个对象的行为都可以获得正确的结果，那这个对象是线程安全的。

2. 使用线程本地存储来解决线程安全问题。

Java 语言中，如果一个变量要被多线程访问，可以使用 volatile 关键字声明它为“易变的”。如果一个变量要被某个线程独享，Java 中就没有类似 C++中__declspec（thread）[3]这样的关键字，不过还是可以通过 java.lang.ThreadLocal 类来实现线程本地存储的功能。每一个线程的 Thread 对象中都有一个 ThreadLocalMap 对象，这个对象存储了一组以 ThreadLocal.threadLocalHashCode 为键，以本地线程变量为值的 K-V 值对，ThreadLocal 对象就是当前线程的 ThreadLocalMap 的访问入口，每一个 ThreadLocal 对象都包含了一个独一无二的 threadLocalHashCode 值，使用这个值就可以在线程 K-V 值对中找回对应的本地线程变量。