多线程

- Java原子操作atomic
  - 类型
    - 标量类
      - AtomicBoolean,
        AtomicInteger,
        AtomicLong,
        AtomicReference
    - 数组类
      - AtomicIntegerArray
        AtomicLongArray
        AtomicReferenceArray
    - 更新器类
      - AtomicLongFieldUpdater,
        AtomicIntegerFieldUpdate,
        AtomicReferenceFieldUpdater
    - 复合变量类
      - AtomicMarkableRefercence
        AtomicStampedReference
  - Synchronized:
    Synchronized主要是用来保证执行顺序，通过synchronized来控制一段代码是否运行并发到达对执行顺序的控制。同时，synchronized还会创建一个内存屏障，内存屏障指令保证了所有的CPU操作结果都会直接刷新到主内存中，从而保证了操作内存的可见性。
  - Volatile:
    告诉线程该变量是易变的，不稳定的。在需要使用到由该关键词修饰的变量时，都需要从主内存中重新获取一次，也就是要同时执行read-load-use，执行后对volatile变量操作后，同时要执行assign-store-write。保证所修饰变量的可见性，一个线程中修改变量的值以后，在其他线程能够看到这个值。
    Volatile保证了变量的可见性，但是不能保证变量的原子性。也就是说，对volatile修饰的变量做出的改变，并不能立刻就写回到主内存中
- 内存管理
  - 线程内存管理：
    1. 每一个线程都拥有一份工作内存，是私有的本地内存，并将共享变量拷贝了一份副本存储在本地内存中。
    2.线程之间通过主内存进行数据交换，所有的变量都存储在主内存中。
    3.线程对变量的所有操作(读取，复制)都必须工作内存中进行，而不能直接读写主内存中的变量
- 数据安全，保持一致性，不会出现脏数据
- 多线程特性
  - 原子性
  - 可见性
  - 有序性
- 线程通信（通过线程的通信保持数据的一致性）
  - 共享内存
    - 线程之间通过读写共享变量的方式进行隐士的通信。线程与进程（本地内存与公共内存），有8中内存操作：
      Lock/unlock
      Read/load/use
      Assign/store/write
    - 内存对象原子操作
      - 1. Java内存模型只需要上述操作按照顺讯执行，但是没有保证这些操作是连续的，可能中间会有一些其他的操作。Read和Load之间，Store和Write之间是可以插入其他指令的。
        2. 不允许Read和Load, Store和Write操作单一存在，必须是成对出现。
        3. 一个新的变量只能在于主内中诞生
        4.如果对一个变量执行lock操作，将会清空工作内存中此变量的值，在执行引擎使用这个变量前需要重新执行load或assign操作初始化该变量的值
        5.对一个变量执行unlock操作之前，必须先把此变量同步到主内中
      - 主内存中的原子操作
        - Lock: 把一个变量标记为一个线程独占的状态
        - Write: 把store操作从工作内存中的一个变量的值传送到住内存的变量中
        - Read: 把一个变量从主内存传输到线程的工作内存中
        - Unlock：把一个处于独占状态的变量释放出来
      - 工作内存中的原子操作
        - Store: 把工作内存的变量的值传递会给主内从中
        - Assign: 把工作内存的一个变量传递给执行引擎
        - Use: 把read操作从主内存得到的变量放入工作内存的变量副本中
        - Load: 把一个从执行引擎接受到的值给工作内存的变量
  - 消息传递
    - 通过显示的方式进行消息的传递。比如调用wait(), notify()等方法进行通信。线程之间没有公共的状态，只能通过明确的发送消息的方式来进行通信。