并发编程安全性问题

- 定义
  - 程序没有按照预期的正确方向执行
- 起因
  - 1.大前提
    - 代码表面上和内部实际上执行的差异
  - 2.小前提：共享变量、多线程环境
  - 3.多个因素
    - 对共享变量的伪原子性操作
    - 编译器指令重拍优化、处理器执行时重排优化
    - 缓存更新顺序和时机（伪重排序）
- 解决方案的基本原则
  - 原子性
    - 保证一组操作完整执行而不被打扰
  - 有序性
    - 保证指令按照预期的顺序执行
  - 可见性
    - 保证多个线程之间共享变量的值在使用时是最新的
- 对应的基本解决方案
  - 互斥
    - 对一组操作进行保护，其他线程想执行这种操作需要进行等待，直到该组操作完成，从而保证原子性
  - 屏障
    - 限定编译器和处理器的重排序规则
  - 缓存同步
    - 给出缓存同步的时机的一套方案
- java提供的底层解决机制
  - 监视器
    - 监视器是基于操作系统的
      - java中的锁，同时保证了原子性、可见性和有序性，且java中提供了众多的锁的优化策略
  - CAS
    - 集检测与更新操作于一体的原子性操作，用于判断线程对某个共享变量的操作的独占性，并执行更新
  - volatile
    - 具备MESI语义以及内存屏障功能，具体表现为对volatile变量的写入会刷新到内存，并且使其他线程对应的缓存行失效，读取时直接从内存中读取，从而获得最新值，同时，在对volatile变量写操作指令后，插入smb，读volatile变量前，插入rmb，从而也具备了内存屏障的功能
  - AQS（CAS、volatile）
    - 使用cas和volatile以及配合队列的数据结构和Thread.park()、unpark等自行实现的java自身的监视器，比内置的监视器灵活，功能更加强大
      - AQS是java中提供的众多线程工具的基础
- java给出了线程安全保证
  - 意义：java给出一套并发安全规则（约定），若程序遵循这套规则，jvm会对程序进行正确的处理，因此程序是多线程安全的，反之则是不安全的，需要更改程序使其符合这套规则
  - 规则1.单线程的as-if-serials语义
  - 规则2.多线程的happens-before规则

试 用 版 XMind:ZEN 试 用 版