volatile的实现原理

# volatile的实现原理---volatile是如何保证共享变量的可见性的?

volatile可以看做是轻量级的synchronized；在多处理器并发中保证了**共享变量的可见性**。

volatile变量的两个语义：

### 保证共享变量的可见性；

### 禁止指令重排序优化；

## 保证共享变量的可见性

面试问题1：**volatile是如何保证共享变量的可见性呢**？

前提基础知识：内存屏障memory barriers、缓冲行cache line、嗅探。

有volatile修饰的共享变量在进行写操作的时候，会多出一行汇编代码，即LOCK前缀的指令，也就是所说的内存屏障。

LOCK前缀指令在多核处理器下会引发两件事情：

### LOCK前缀指令会将当前处理器缓存行的数据**回写**到系统内存<主内存>；

### 一个处理器的缓存回写到内存的操作会导致其他处理器里缓存了该内存地址的数据无效。

嗅探技术：处理使用嗅探技术保证它的内部缓存、系统内存和其他处理器的缓存的数据在总线上保持一致。

在多处理器下，为了保证各个处理器的缓存一致性，采用的是缓存一致性协议。

MESI协议(修改、独占、共享、无效)控制协议。

锁定总线==>锁定缓存行；

锁定总线的开销比较大，锁定缓存行的开销小，再利用缓存一致性协议实现**缓存一致性**。面试问题2：volatile变量在各个线程的工作内存中不存在一致性问题，那么为什么还说基于volatile的运算在并发下是不安全的呢?

因为Java里面的运算并非原子操作，导致volatile变量的运算在并发下一样是不安全的。

举个例子：++操作是一个复合操作，包括int temp = get(); temp += 1; set(temp);

再get()操作是获取到最新的值，但是在get()操作之后与set(temp)之间，volatile的值可能被其他线程改变了，但是执行引擎看不到这种改变，因此仍然是不安全的。

所以：对于单一的读或写操作具有原子性，但是对于复合操作不具有原子性。

## 禁止指令重排序

禁止指令重排序的实现，是基于内存屏障的作用，内存屏障就是LOCK前缀指令。

内存屏障分成4种：

读读LoadLoad

写写StoreStore

读写LoadStore

**写读StoreLoad<全能型>**

# synchronized的实现原理

synchronized的实现原理：

代码同步块的实现是通过monitorenter和monitorexit指令实现的。

同步方法是通过另外一种方法实现的。