第三章 对象的共享

#### 一．可见性

重排序

1. 概念

在没有同步的情况下，编译器、处理器以及运行时等都可能对操作的执行顺序进行一些意想不到的调整。在缺乏足够同步的多线程程序中，要想对内存的执行顺序进行判断，几乎无法得出正确的结论。

2. 应用场景

Java内存模型和CPU允许对操作进行重排序，并将数值缓存在寄存器（Java内存模型）处理器特定的缓存（CPU）中。

3. 优点

重排序可以使得JVM能充分的利用现代多核处理器的强大性能。

非原子的64位操作

1. 概念

Java内存模型要求，变量的读取和写入操作都必须是原子操作，但对于非volatile类型的long和double变量，JVM允许将64位的读操作或写操作分解为两个32位的操作。当读取一个非volatile的long变量时，如果对变量的读写操作在不同的线程中执行，那么很可能会读取到某个值的高32位和另一个值的低32位。

所以不考虑数据失效问题，在多线程程序中使用共享且可变的long和double等类型的变量也是线程不安全的，除非用关键字volatile来声明它们，或者用锁保护起来。

加锁与可见性

1. 加锁的含义不仅仅局限于互斥行为，还包括内存可见性。为了确保所有线程都能看到共享变量的最新值，所有执行读操作或者写操作的线程都必须在同一个锁上同步。

Volatile

1. 禁止重排序

当把变量声明为volatile时，编译器与运行时会注意到这个变量是共享的，因此不会将改变量上的操作与其它内存操作进行重排序。

2. 保证可见性

Volatile变量不会被缓存在寄存器或者其它处理器不可见的地方，因此在读取volatile类型的变量时总会返回最新写入的值。

3. 轻量级

Volatile变量不会执行加锁操作，因此它是一种比synchronized更加轻量级的同步机制。

4. 使用场景

仅当volatile变量能简化代码的实现以及对同步策略的验证时，才应该使用它们。

当且仅当满足以下条件时，才应该使用volatile变量：

* 对变量的写入操作不依赖变量的当前值，或者你能确保只有单个线程更新变量的值。
* 该变量不会与其他状态变量一起纳入不变性条件中。
* 在访问变量时不需要加锁。

5. 与加锁机制的区别

加锁机制既可以确保可见性又可以确保原子性，而volatile变量只能确保可见性。

发布与逸出

1. 发布

发布一个对象是指，使对象能够在当前作用域之外的代码中使用。

2. 逸出

当某个不应该被发布的对象被发布时，称为逸出。

3. 安全的对象构造过程

不要再构造过程中使this逸出。

不要在构造函数中注册事件监听器或者启动线程，如果非要这么做可以使用一个私有的构造函数和一个公共的工厂方法。