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，因此不会将该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。

#### 二. 线程封闭

1. 概念

当访问共享可变数据时，需要使用同步。一种避免使用同步的方式是不共享数据，如果仅在单线程内访问数据就不需要使用同步。这种技术叫线程封闭。例如，JDBC的Connection对象就使用了线程封闭技术。

栈封闭

1. 概念

栈封闭是线程封闭的一种特例，在栈封闭中，只能通过局部变量才能访问对象。

ThreadLocal类

1. ThreadLocal为每一个线程维护一个变量副本。

2. 使用场景

用于防止对可变的单实例变量或全局变量进行共享。

3. 缺点

增加耦合

#### 三. 不变性

不可变对象

1. 概念

如果一个对象在创建后不能修改，那么这个对象就是不可变对象。

2. 安全性

不可变对象一定是线程安全的。

3. 满足以下条件，对象才是不可变的

* 对象的所有域都是final类型
* 对象是正确创建的（在对象的创建期间，this引用没有逸出）
* 对象创建以后其状态就不能修改

final域

1. final类型的域是不能修改的，final域能保证初始化过程的安全性，从而可以不受限制的访问不可变对象，并在共享这些对象时不需要使用同步。

2. “除非需要更高的可见性，否则应该将所有的域都声明为私有域”是一个良好的编程习惯，“除非需要某个域是可变的，否则应将其声明为final域”也是一个良好的编程习惯。

3. 使用volatile来发布不可变对象。

#### 四. 安全发布

不正确的发布：正确的对象被破坏

1. 在构造函数中设置域值可能会导致不正确的发布。

尽管在构造函数中设置的域值是第一次向这些域中写入的值，因此不会有“更旧的”值被视为失效值。但Object的构造函数会在子类构造函数运行之前先将默认值写入所有的域。因此某个域的默认值可能视为失效值。

不可变对象与初始化安全性

1. 任何线程都可以在不需要额外同步的情况下安全的访问不可变对象，即使在发布这些对象时没有使用同步。

2. 安全发布的常用模式

* 在静态初始化函数中初始化一个对象的引用。
* 将对象的引用保存到volatile类型的域或者AtomicReferance对象中。
* 将对象的引用保存到某个正确构造对象的final类型域中。
* 将对象的引用保存到一个由锁保护的域中。

3. 在线程安全容器内部的同步（满足上述最后一条）

将键放入，Hashtable，ConcurrentMap中，或者把元素放入Vector，CopyOnWriteArrayList中，把元素放入BlockingQueue或者ConcurrentLinkedQueue中。

4. 要发布一个静态构造的对象，最简单和最安全的方式是使用静态的初始化器。

public static Holder holder = new Holder(42);

静态初始化器由JVM在类的初始化阶段执行。由于JVM内部存在着同步机制，因此通过这种方式初始化的任何对象都可以被安全的发布。

事实不可变对象

1. 概念

如果对象从技术上来看是可变的，但其状态在发布后不会再改变，那么这种对象称为不可变对象。

2. 安全性

在没有额外的同步的情况下，任何线程都可以安全的使用被安全发布的事实不可变对象。

可变对象

1. 对于可变对象，不仅发布的时候需要使用同步，而且在每次对象访问的时候同样需要使用同步。

2. 总结

* 不可变对象可以通过任何方式发布
* 事实不可变对象必须通过安全方式来发布
* 可变对象必须使用安全方式发布，并且必须是线程安全的或者由某个锁保护起来。

安全的共享对象

1. 线程封闭

2. 只读共享

3. 线程安全共享

4. 保护对象