第二章 线程安全性

2.1 线程安全

1.*无状态对象一定是线程安全的。*线程访问无状态对象的行为不会影响其他线程操作的正确性，例如大多数servlet就是无状态的，所以servlet一般就是线程安全的，只有处理请求时servlet需要保存信息，才回引发线程安全问题。

2.2 原子性

1．原子性。确保一系列的操作是原子操作，才能保证线程安全。例如对于计数类操作，多使用AtomicLong等原子变量，就可以保证线程安全

2.3 加锁机制

1. java内置锁：同步代码块（Synchronized Block）包括两部分：一个是锁的对象引用，一个是锁保护的代码块。用synchronized修饰的方法就是锁保护起来的代码块，同步代码块的锁就是方法调用的对象。静态synchronized方法以class对象作为锁。

2.重入：一个线程试图获取自己持有的锁的时候，会发生重入，所以请求会成功，不会死锁。

2.5 活跃性与性能

当执行时间较长的计算或者无法完成快速操作的时候，例如网络io或者控制台io，一定不要持有锁。

第三章 对象的共享

3.1 可见性

1. java内存模型要求，变量的读取和写入都必须是原子操作。对于非volatile修饰的double和long类型，jvm将64位的读写分成两个32位的原子操作，所以是线程不安全的，除非volatile声明，或者锁保护起来。

加锁机制既可以保证可见性又可以保证原子性，volatile只能保证可见性

2.volatile使用条件：

1.对变量的写入操作不依赖于变量的当前值

2.该变量不会与其他变量纳入不变形条件中

3.在访问变量时不需要加锁

3.2 发布和逸出

发布：让对象可以在当前作用域之外的代码中访问到。例如将对象的引用保存到其他代码可以访问的地方，或非私有方法中返回该引用，或将引用传递到其他类

逸出：某个不应该发布的类被发布了。

注意：不要在构造函数中使this引用逸出

3.3 线程封闭

1.维持线程封闭的更规范方法是ThreadLocal。

3.4 不变性

1.不可变对象一定是线程安全的。不可变对象需要满足的条件：

1.对象创建以后就不能再修改

2.对象的所有域都是final类型

3.对象是正确创建的（在对象的创建期间，this引用没有逸出）

第十一章、性能与可伸缩性

11.3 引入线程的开销

对于为了提升性能而引入的线程来说，并行带来的性能提升必须超过并发导致的开销。

1、上下文切换

如果可运行线程大于cpu数量，操作系统会将正在运行的线程进行调度，这将导致一次上下文切换：保存当前运行线程的执行上下文，将新调度进来的线程的执行上下文设置为当前上下文。一般消耗几微秒

2、内存同步

某个线程中的同步可能会影响其他线程的性能。同步会增加共享内存总线上的通信量，因为总线带宽有限，而且所有处理器都共享这条总线，所以如果多个线程竞争同步带宽，性能会下降

3、阻塞

线程阻塞被挂起过程中包含两次额外的上下文切换，被阻塞的线程在执行时间片还未用完就交换出去，随后获得锁或者资源可用时再次被切换回来。（由于锁竞争而导致阻塞时，线程在持有锁时存在一定的开销：当释放锁时必须告诉操作系统恢复运行阻塞的线程）

11.4 减少锁的竞争

三种方式减低锁的竞争程度：

减少锁的持有时间

降低锁的请求频率

使用带有协调机制的独占锁，这些锁允许更好的并发性

1.缩小锁的范围：在真正需要加锁的地方加锁

2.减小锁的粒度：锁分解，采用相互独立的锁保持自己的状态

3.锁分段：对锁分解的扩展，将一组独立对象的锁进行分解。concurrentHashMap利用的就是锁分段技术，包含16个锁的数组。

4.替代独占锁的方法，比如readWriteLock，静态计数器，序列发生器，或者对列表数据结构中头结点的引用

5.检测cpu利用率：vmstat，mpstat命令，或者Windows的perfmon.exe

6.减少上下文的开销：例如对于记录日志，可以将日志进行异步打印

第十二章、并发程序的测试

性能可以通过多个方面来衡量

吞吐量：一组并发任务中已完成任务所占比例

响应性：请求从发出到完成之间的时间（也成为延迟）

可伸缩性：在增加更多资源的情况下（通常指CPU），吞吐量的提升情况

第十三章、显示锁 lock

Synchronized是jvm的内置属性，所以他能执行一些优化，例如对线程封闭的锁对象进行锁消除优化，通过增加锁的粒度来消除内置锁的同步，而ReentrantLock是基于代码层面的，就不能实现这些优化。

扩展：

1.Synchronized缺陷

Synchronized修饰的代码块只有两种情况可以释放锁

1）获取锁的线程执行完代码块，然后释放锁

2）线程执行发生异常，此时JVM会让线程自动释放锁

如果这个线程被阻塞了但是又没有释放锁，其他线程只能等待，影响执行效率，lock可以更好的控制。

使用lock时需要注意：

1）lock时Java语言定义的，而synchronized是jvm内置特性

2）lock需要用户手动释放锁，所以一般要try{…} finally{…}