并发编程

1. 方法或者类的无状态

一个方法或者一个类既不包含任何域,也不包含对其它类中域的引用就称为无状态.无状态对象一定是线程安全的.该类可能有字段,但他们一定是编译时常量(static final).

1. 竞态条件

当某个计算的正确性取决于多个线程的交替执行时序时就会发生竞态条件,比如通过一个可能失效的观测结果决定下一步的动作.

1. volatile变量:

只能保证可见性,不能保证原子性.仅当能简化代码的实现以及对同步策略的验证时,才应该使用.比如检查某个状态标记以判断是否退出循环.

使用场景,满足以下条件:

1. 对变量的写入操作不依赖变量的当前值(++操作不行),或者确保只有单个线程更新变量的值.
2. 该变量不会与其他状态变量一起纳入不变性条件中.
3. 在访问变量时不需要加锁.

不可变对象可以通过任意机制来发布;

事实不可变对象(内部实现了同步机制)必须通过安全方式来发布;

可变对象必须通过安全方式发布,并且必须实现线程安全.

1. java监视器模式

将可变对象进行封装,并由对象自己的内置锁来实现线程安全.

当为现有的类添加一个原子操作时,可以通过组合(implements)添加锁方法.

1. 对象逸出

造成对象逸出的原因通常是因为对象的发布超出了既定的作用域.

基础构建模块

ConcurrentModificationException

Queue/BlockingQueue 生产者消费者模式

ConcurrentHashMap

CopyOnWriteArrayList

闭锁 CountDownLatch

FutureTask

死锁:

避免产生锁顺序死锁,确保线程在获取多个锁时采用一致的顺序.

在持有锁时调用外部方法,则可能发生死锁.

使用开放调用,采用封装机制提供线程安全的方法.

可伸缩性:

可伸缩性指的是当增加计算资源时(如CPU,内存,存储容量或I/O带宽),程序的吞吐量或者处理能力能相应的增加.

在所有并发程序中都包含一些串行部分.需要同步的比例越大,并发的效率越低.

提高可伸缩性:

1. 减少锁的持有时间;2降低锁的力度,也就是保证串行的代码量最低;3.采用非独占的锁或非阻塞的锁代替独占锁.

锁分解或者锁分段技术.

显式锁:

Lock ReentrantLock 对于定时以及轮询或者可中断的锁获取操作,公平队列以及非块结构的锁时可以代替synchronized,在性能方面其实相差不大.读写锁允许多个读线程并发访问被保护的对象,可以提高程序的可伸缩性.