Lock相关: <https://blog.csdn.net/nalanmingdian/article/details/77800355>

# JUC 简介

在 Java 5.0 提供了 java.util.concurrent(简称JUC)包,在此包中增加了在并发编程中很常用的工具类，用于定义类似于线程的自定义子系统,包括线程池,异步 IO 和轻量级任务框架;还提供了设计用于多线程上下文中的 Collection 实现等;

# volatile 关键字

volatile 关键字: 当多个线程进行操作共享数据时,可以保证内存中的数据是可见的，不会引起线程上下文的切换和调度;相较于 synchronized 是一种较为轻量级的同步策略;

volatile 不具备"互斥性";

volatile 不能保证变量的"原子性";

volatile是一种“轻量级的锁”，它能保证锁的可见性，但不能保证锁的原子性；

使用场景：对变量的写操作不依赖当前值；该变量没有包含在具有其他变量的不变式中。

底层实现原理与java内存模型 相关。

# i++ 的原子性问题

i++的操作实际上分为三个步骤: "读-改-写";

原子性: 就是"i++"的"读-改-写"是不可分割的三个步骤;

原子变量: JDK1.5 以后, java.util.concurrent.atomic包下,提供了常用的原子变量;

原子变量中的值,使用 volatile 修饰,保证了内存可见性;

CAS(Compare-And-Swap) 算法保证数据的原子性;

# CAS 算法

CAS(Compare-And-Swap) 算法是硬件对于并发的支持,针对多处理器操作而设计的处理器中的一种特殊指令,用于管理对共享数据的并发访问;

CAS 是一种无锁的非阻塞算法的实现;

CAS 包含了三个操作数:

需要读写的内存值: V

进行比较的预估值: A

拟写入的更新值: B

当且仅当 V == A 时, V = B, 否则,将不做任何操作;

# 并发容器类

## ConcurrentHashMap

ConcurrentHashMap 同步容器类是 Java5 增加的一个线程安全的哈希表;介于 HashMap 与Hashtable 之间;

内部采用"锁分段"机制替代Hashtable的独占锁,进而提高性能;

此包还提供了设计用于多线程上下文中的Collection（收集）实现:

ConcurrentHashMap，ConcurrentSkipListMap，ConcurrentSkipListSet， CopyOnWriteArrayList 和CopyOnWriteArraySet;

当期望许多线程访问一个给定collection时,ConcurrentHashMap通常优于同步的HashMap;

ConcurrentSkipListMap通常优于同步的TreeMap;

当期望的读数和遍历远远大于列表的更新数时, CopyOnWriteArrayList优于同步的ArrayList;

## CountDownLatch(闭锁)

CountDownLatch是一个同步辅助类,在完成一组正在其他线程中执行的操作之前,它允许一个或多个线程一直等待;

# 创建执行线程的方式三

相较于实现 Runnable 接口的方式,实现 Callable 接口类中的方法可以有返回值,并且可以抛出异常;

## 创建和启动线程的步骤

1.创建Callable接口的实现类，并实现call()方法，该call()方法将作为线程执行体，且该call()方法没有返回值，再创建Callable实现类的实例。（从java8开始，可以直接使用Lambda表达式创建Callable对象）。

2.使用FutureTask类来包装Callable对象，该FutureTask对象封装了该Callable对象的call()方法的返回值。

3.使用FutureTask作为Thread对象的target创建并启动新线程。

4.调用FutureTask对象的get方法来获得子线程执行结束后的返回值。

# 同步锁(Lock)

## Lock 接口

1.Lock 接口, 位于 java.util.concurrent.locks 包中, 使用该接口需要导包.

2.Lock 接口的出现替代了同步代码块或者同步函数, 因为同步代码块对于锁的操作(获取或释放)是隐式的。Lock 接口将同步的隐式锁操作变成显式锁操作。同时,更为灵活, 可以在一个锁上加上多个监视器.

3.Lock 接口中的方法:

lock(): 获取锁

unlock(): 释放锁, 这个动作是必须要完成的, 所以通常需要定义在 finally 代码块中

### 几种Lock锁

ReentrantLock，它是一个：默认非公平但可实现公平的，悲观，独享，互斥，可重入，重量级锁。

ReentrantReadWriteLocK，它是一个，默认非公平但可实现公平的，悲观，写独享，读共享，读写，可重入，重量级锁。

如果使用 synchronized ，如果A不释放，B将一直等下去，不能被中断

如果 使用ReentrantLock，如果A不释放，可以使B在等待了足够长的时间以后，中断等待，而干别的事情

## ReentrantLock获取锁定有三种方式：

lock(), 如果获取了锁立即返回，如果别的线程持有锁，当前线程则一直处于休眠状态，直到获取锁

tryLock(), 如果获取了锁立即返回true，如果别的线程正持有锁，立即返回false

tryLock(long timeout,TimeUnit unit)， 如果获取了锁定立即返回true，如果别的线程正持有锁，会等待参数给定的时间，在等待的过程中，如果获取了锁定，就返回true，如果等待超时，返回false；

lockInterruptibly:如果获取了锁定立即返回，如果没有获取锁定，当前线程处于休眠状态，直到获取锁定，或者当前线程被别的线程中断

**按照其性质分类**

公平锁/非公平锁

乐观锁/悲观锁

独享锁/共享锁

互斥锁/读写锁

可重入锁

**按照设计方案来分类**

自旋锁/自适应自旋锁

锁粗化/锁消除

偏向锁/轻量级锁/重量级锁

分段锁

## Condition 接口

1.Condition 接口的出现替代了 Object 类中的 wait(), notify(), notifyAll()方法,将这些监视器方法单独进行了封装, 变成 Condition 监视器对象, 可以与任意锁进行组合.

2.常用方法:

await(): 让线程处于冻结状态

signal(): 唤醒一个等待线程

signalAll(): 唤醒所有等待线程

3.格式:

Condition c1 = lock.newCondition(); // 新建一个监视器对象

## wait() 和 sleep() 的区别

1.wait() 可以指定时间也可以不指定时间

sleep() 必须指定时间.

2.在同步中, 对 CPU 的执行权和锁的处理不同

wait(): 释放执行权, 释放锁

sleep(): 释放执行权, 不释放锁

## 停止线程

1.run() 方法结束

2.怎么控制线程的任务结束呢?

任务中都会有循环结构, 只要控制住循环就可以结束任务.

控制循环通常就用定义标记(条件)来完成.

### 如果线程处于冻结状态, 无法读取标记, 如何结束呢?

可以使用 interrupt() 方法将线程从冻结状态强制恢复到运行状态, 让线程具备 CPU 的执行资格.

该强制动作会发生 InterruptedException, 需要处理.

## 线程类的其他方法

1.setDaemon(boolean b) : 将该线程标记为守护线程(后台线程).

2.join() : 临时加入一个线程运算时, 可以使用 join() 方法, 需要处理 InterruptedException

3.toString() : 返回该线程的字符串表示形式, 包括线程名称, 优先级和线程组.

优先级: 获取 CPU 执行权的机率, 分为 1~10个数字, 其中:

最高优先级: MAX\_PRIORITY , 代表数值 10

最低优先级: MIN\_PRIORITY , 代表数值 1

默认优先级: NORM\_PRIORITY , 代表数值 5

4.yield(): 暂停当前正在执行的线程对象, 并执行其他线程.

# 线程池

线程池提供了一个线程队列,队列中保存着所有等待状态的线程;

避免了创建与销毁线程的额外开销,提高了响应速度;

线程池的体系结构

java.util.concurrent.Executor: 负责线程的使用和调度的根接口;

ExecutorService: 子接口,线程池的主要接口;

ThreadPoolExecutor: 线程池的实现类;

ScheduledExecutorService: 子接口,负责线程的调度;

ScheduledThreadPoolExecutor: 继承了线程池的实现类,实现了负责线程调度的子接口;

工具类: Executors

ExecutorService newFixedThreadPool(): 创建固定大小的线程池;

ExecutorService newCachedThreadPool(): 缓存线程池,线程池中线程的数量不固定,可以根据需求自动更改数量;

ExecutorService newSingleThreadExecutor(): 创建单个线程池, 线程池中只有一个线程;

ScheduledExecutorService newScheduledThreadPool(): 创建固定大小的线程,可以延时或定时的执行任务;

# Fork/Join 框架