Java 多线程编程核心技术 笔记

作者：西瓜1994
时间：2018/10/12 09:20
标签： Java

外练互斥,内修可见。

1：currentThread() 方法可返回代码正在被哪个线程调用的信息
使用方法：Thread.currentThread.getName()

2:isAlive() 功能是判断当前线程是否处于活动状态
使用方法：Thread.currentThread.isAlive()

3:sleep() 作用是在指定的毫秒数内让当前“正在执行的线程”休眠（暂停执行），sleep 会让出cpu执行时间片，sleep不释放锁。
使用方法：Thread.sleep(2000)

4：getId()方法的作用是取得线程的唯一标识
使用方法：Thread.currentThread.getId();

5:停止线程，在java中有三种方法可以终止正在运行的线程
5-1：使用退出标志，使线程正常退出，也就是当run方法执行完成后线程终止。
5-2：使用stop方法强行终止线程，但是不推荐这个方法，因为stop和suspend以及resume一样，都是过期作废的方法，使用它们会导致线程不安全。
5-3：使用interrupt方法中断线程

6：interrupted() 测试当前线程是否已经中断，执行后具有将状态标志清除为flase的功能
使用方法;thread.interrupted() ps：返回boolean类型，此时将返回true,代表中断成功。

7：isInterrupted() 测试线程Thread对象是否已经是中断状态，但不清除状态标志。
使用方法：thread.isInterrupted() ps：返回true,代表线程已经停止。

8：yield() 方法的作用是放弃当前的cpu资源，让给其他的任务去占用cpu的执行时间，但是放弃的时间不确定，有可能刚刚放弃，马上又获得cpu时间片，这可能是由于自身线程的优先级较高。
使用方法：Thread.yield()

9:线程优先级
设置线程优先级使用方法：Thread.currentThread.setPriority(6)

线程优先级具有继承性：比如A线程启动了B线程，那么B线程的优先级和A线程的优先级一样。
线程优先级具有规则性：高优先级的线程总是大部分先执行完，但不代表高优先级的线程要全部先执行完。而且当优先级的差距很大时，谁先执行完和代码的调用顺序无关。

线程优先级具有随机性：线程的优先级具有随机性，优先级搞得线程不一定每一次都先执行完。

10：守护线程

在java线程中，有两种线程一种是守护线程，另一种是用户线程也就是非守护线程。

守护线程是一种特殊的线程，它的特性有陪伴的含义，当进程中不存在非守护线程时，守护线程自动销毁。

java中典型的守护线程就是垃圾回收线程（GC），当进程中没有非守护线程时，垃圾回收线程也没有存在的必要了，自动销毁。

使用方法：thread.setDaemon(true)

.

11:synchronized 取得的锁都是对象锁，而不是把一段代码或方法当做锁，哪个线程先执行带synchronized关键字的方法，

哪个线程就持有该方法所属对象的Lock，那么其他线程只能呈等待状态，但是此时其他线程可以访问被锁对象的非同步方法，

前提是多个线程访问的是同一个对象。如果多个线程访问不同的对象，那么JVM就会创建多个锁。

12：synchronized 可重入锁说明：

可重入锁的概念是：自己可以再次获取自己的内部锁，比如有一个线程获得了某个对象的锁，此时这个线程还没有释放这个对象的锁，那么它再次想要获取这个对象的锁还是可以获取到的。

如果是不可重入锁的话，就会造成死锁。

13：当一个线程出现异常时，锁会自动释放。其他线程就可以获取到锁，继续执行。

14：synchronized 不具有继承性，也就是说父类的方式是synchronized声明的，子类继承并实现该方法是不带同步功能的，所以要想实现父类的同步功能，在子类中的方法中也要加上 synchronized 关键字。

15：当使用synchronized时，最好不要同步整个方法，因为同步整个方法效率太低，最好是同步代码块，在真正需要同步的代码块上加入synchronized关键字

16：synchronized 应用在static静态方法上时，是对当前的java文件对应class类进行加锁,此时多个线程访问多个相同对象的静态方法 也是需要排队获取class类的锁的。

17：synchronized 不要使用String作为锁对象，因为String 常量池的影响

例如：

```
public static void main(String args[]){
    String a="AA";
    String b="AA";
    System.out.println(a==b);
}
```

结果是输出 true，因为AA是在常量池中，所以变量a和变量b指向的都是常量池中的AA。

所以当使用synchronized锁住String对象时，会造成线程持有相同的锁，导致只有一个线程一直在运行，其他线程一直获取不到锁。

18：synchronized 包含两个特性：互斥性和可见性。

19：volatile 关键字增加了实例变量在多个线程之间的可见性，但volatile不支持原子性

volatile和synchronized区别

1：关键字volatile是线程同步的轻量级实现，所以vloatile性能要比synchronized好，并且volatile只修饰变量，而synchronized可以修饰方法 以及代码块。

2：多线程访问volatile不会发生阻塞，而sunchronized会阻塞

3：volatile能保证数据的可见性，但不能保证原子性，而sunchronized可以保证原子性，也可以间接的保证可见性，因为它会将私有内存和公共内存的数据做同步。

20：使用原子类进行i++操作，例如AtomicInteger

使用方法：

```
private AtomicInteger count=new AtomicInteger(0);
public  void add(String args[]){
    for(int i=0;i<100;i++){
        count.incrementAndGet();
    }
}
```

21:原子类也不能保证线程一定安全，使用原子类修饰的变量可以保证原子性，但是方法和方法之间的调用，不是原子的，很有可能调用顺序出错。

所以应该在方法上加入synchronized关键字配合原子类。

22：wait()方法的作用是使当前执行代码的线程进行等待，wait()方法是Object类的方法，该方法用来将当前线程放到“预执行队列”中，

wait()方法在调用前，需要获取该对象的对象级别锁，所以它必须在同步方法或者同步代码块中调用。

23：notify()方法的作用是释放该对象的锁并通知那些呈wait状态的其他线程，对其发出通知，此时线程规划器会随机挑选一个wait的线程，并使它等待获取该对象的对象锁。

再调用notify（ ）后，并不会立即执行，wait()方法的下面代码。

而是需要等到执行notify方法的同步代码块中的代码全部执行完以后释放了该对象的锁，此时才能执行wait（ ）方法下面的代码。

24:用一句话总结wait和notify：wait是线程停止运行，并释放锁。而notify会随机使一个停止的线程继续运行。

25：notifyAll()方法唤醒 同一共享资源的“全部”线程，使其从等待状态退出，进入到可运行状态。根据不同虚拟机的实现，有可能是优先级最高的那个线程先获取到锁先执行，也有可能是随机执行。

26：wait(long) 方法 是等待某一时间内是否有线程对锁进行唤醒，如果超过这个时间则自动唤醒。

27：join()方法作用是 使线程按照给定的顺序运行。 join具有使线程排队的作用，有些类似同步的运行效果，

join与synchronized的区别是：join内部使用wait()方法进行等待，而synchronized关键字则使用“对象监视器”原理作为同步。

例如：

```
public static void main(String args[]){
    MyThread threadTest=new MyThread();
    threadTest.start();
    threadTest.join();
    System.out.println("threadTest 线程运行完后，我才会打印这句话")；
}
```

28：Threadlocal 第一次调用get()时值为null的解决办法， 新建一个类，继承ThreadLocal 并实现initialValue()方法，返回默认值。

29：当使用ThreadLocal时，要在子线程中获取父线程使用的ThreadLocal的值时，需要使用InheritableThreadLocal这个类

InheritableThreadLocal 和 ThreadLocal 最大的区别就是： 使用ThreadLocal修饰的变量，子类不能继承父类的变量值。 而使用InheritableThreadLocal 可以。

当子类需要改变被InheritableThreadLocal修饰的变量值时，只需要重写childValue（ ）方法。

需要注意的是 使用InheritableThreadLocal时，如果子线程在取得值得同时，主线程将InheritableThreadLocal中的值进行更改，那么子线程取到的值还是旧值。

30： ReentrantLock lock()加锁 unlock()释放锁 类似于Synchronized关键字。不过ReetrantLock的功能和效率要比Synchronized强大

31：Condition 对象监视器，主要是配合ReetrantLock 实现ReetrantLock 的唤醒和等待。

```
Condition condition=new ReetrantLock.newCondtion();
```

condition.await() 等待 相当于Object类中的 wait()

condition.signal() 随机唤醒一个await状态的线程 相当于Object类中的 notify()

condition.signalAll() 唤醒全部 相当于Object类中的notifyAll()

32:ReentrantLock 可以设置公平锁或是非公平锁

公平锁指的是：线程获取锁的顺序是按照线程加锁的顺序来分配的，即先进先出FIFO的顺序。性能不如非公平锁。

非公平锁就是一种获取锁的抢占机制，随机获取锁，可能优先级高的获取锁的几率会大一些。这样有的线程可能一直获取不到锁，结果也就是不公平了。

ReetrantLock reetrantLock=new ReetrantLock(true)； ReetrantLock的构造函数中，传入true就是公平锁，不传默认为非公平锁

33：ReentrantLock的 getHoldCount()方法的作用是 查询当前线程保持此锁定的个数，也就是调用lock()方法的次数。

34：ReentrantLock的 getQueueLength() 方法的作用是 返回正在等待获取锁的线程估计数，

35：ReentrantLock的 getWaitQueueLength() 方法的作用是返回执行同一个condition（对象监视器）的await()方法的估计数。 比如有5个线程调用了同一个ReetrantLock对象的await（ ）方法，

那么getWaitQueueLength()返回5

36：ReentrantLock的 hasQueuedThread(Thead thread) 方法作用是 查询指定的线程是否正在等待获取锁

ReentrantLock的 hasQueuedThreads() 方法的作用是 查询是否有线程正在等待获取锁

37：ReentrantLock的 hasWaiters(Condition condition) 的作用是查询是否有线程正在等待与此锁定有关的condition条件。

38：ReentrantLock的 isFair() 的作用是判断是不是公平锁

39：ReentrantLock的 isHeldByCurrentThread() 的作用是 查询当前线程是否保持此锁定

40：ReentrantLock的 lockInterruptibly 的作用是 如果当前线程未被中断，则获取锁定，如果已经被中断则出现异常。

41：ReentrantLock的 tryLock() 仅在调用时 当前的锁未被另一个线程保持的情况下，才会获取该锁定。

41：ReentrantLock的 tryLock(long timeout,TimeUnit unit) 的作用时，如果锁定在给定等待时间内没有被另一个线程保持，且当前线程未被中断，则获取该锁定。

42：读写锁 ReentrantReadWriteLock 类

读写锁表示有两个锁，一个是读操作相关的锁 共享锁 ，一个是写操作相关的锁 排他锁， 多个读锁之间不互斥，读锁与写锁互斥，写锁与写锁互斥。

ReentrantReadWriteLock lock =new ReentrantReadWriteLock();

lock.readLock.lock(); 获取读锁

lock.writeLock.lock(); 获取写锁
