Java 多线程

版本：2018/8/29-1(23:59)

synchronized

1、synchronized是什么?

> 1. Java中的一个关键字，Java语言内置的特性。
> 2. synchronized修饰的代码块，当某一线程获取锁并且执行该代码段时，其他线程只能一直等待，等待该锁被释放。

2、使用synchronized时，线程如何才能释放锁?

> 1. 线程执行完毕，释放锁。
> 2. 线程执行异常，JVM会让线程释放锁。

3、synchronized的问题

> 1. 如果获取锁的线程出现了阻塞(IO操作、调用sleep)，却没有释放锁，会导致等待线程一直等待，影响效率。
> 2. 一种场景：当多线程读写文件时，读操作和写操作会发生冲突，写操作和写操作会冲突，但是读操作和读操作不会冲突。synchronized会导致一个线程在读取时，其他线程只能等待且无法进行读取操作。

Lock

1、Lock是什么？Lock的特点?

> 1. jdk 1.5后新增
> 2. java.util.concurrent.locks中提供的Lock接口。
> 3. 不是Java语言内置的特性。
> 4. 通过Lock可以获知线程是否成功获取到锁。
> 5. 具有一种机制可以不让等待的线程无限等待下去。
> 6. 能处理读写锁的场景。
> 7. 需要手动去释放锁

2、Lock和synchronized的区别?(6)

> 1. synchronized是关键字；Lock是接口
> 2. synchronized无法处理多个线程能同时读操作的场景；Lock可以处理
> 3. synchronized无法知道线程有没有成功获取到锁；Lock可以知道
> 4. synchronized不需要手动释放，执行完系统会自动释放；Lock必须要手动释放，否则可能会出现死锁。
> 5. synchronized无法中断等待锁的线程；Lock可以通过 `lockInterruptibly()` 和 `interrupt()` 中断正在等待锁的线程。
> 6. 使用方式：synchronized是修饰整个方法或者代码块；Lock可以在任何地方调用lock()和unlock()
> 7. 性能上：如果竞争资源非常激烈时，Lock的性能远远优于synchronized。

3、Lock接口有哪些方法(6种)?

```
public interface Lock {
  // 1、获取锁或者进行等待
    void lock();
  // 2、获取锁或者进行等待，且该等待的时候可以被中断---退出现成的等待状态
    void lockInterruptibly() throws InterruptedException;
  // 3、尝试获得锁，该方法不会阻塞，获得锁就返回true，没有获得锁就返回false
    boolean tryLock();
  // 4、相对于tryLock，会有一个等待时间。等待一阵子且没有获得锁，才会返回false。
    boolean tryLock(long time, TimeUnit unit) throws InterruptedException;
  // 5、释放锁
    void unlock();
  // 6、获取到条件变量
    Condition newCondition();
}
```

4、Lock的lock方法实例

```
Lock lock = ...;
lock.lock();
try{
    //处理任务
}catch(Exception ex){

}finally{
    lock.unlock();   //释放锁
}
```

5、Lock的tryLock()方法

> 1. 尝试获得锁，如果获得锁，返回true。
> 2. 尝试获得锁，获取失败(锁被其他线程占有)，返回false。
> 3. 该方法不会阻塞

```
Lock lock = ...;
if(lock.tryLock()) {
     try{
         //处理任务
     }catch(Exception ex){

     }finally{
         lock.unlock();   //释放锁
     }
}else {
    //如果不能获取锁，则直接做其他事情
}
```

6、Lock的lockInterruptibly()作用

> 解释： 线程A已经获得了锁，此时线程B调用lockInterruptibly去获取锁，线程B会进入等待状态。
> 调用 `threadB.interrupt()` 会中断线程B的等待。
> 注意：如果线程B已经获得了锁，此时调用 `threadB.interrupt()` 无法中断线程B。

```
public void method() throws InterruptedException {
    lock.lockInterruptibly();
    try {
     //.....
    }
    finally {
        lock.unlock();
    }
}
```

ReentrantLock

7、ReentrantLock是什么?

1. Lock的实现类
2. 可重入锁

ReadWirteLock

8、ReadWirteLock是什么?

1. 一个接口
2. 只定义了两个方法：读锁和写锁

```
public interface ReadWriteLock {
    // 1、读锁
    Lock readLock();
    // 2、写锁
    Lock writeLock();
}
```

ReentrantReadWriteLock

9、ReentrantReadWriteLock是什么?

1. 实现了ReadWirteLock接口。
2. 线程A获取了读锁，线程B去获取读锁能直接获取，并且执行读取操作。线程C去获取写锁，就会被阻塞。
3. 线程A获取了写锁，此时线程B获取读锁会被阻塞，线程C获取写锁也会被阻塞。

10、ReentrantReadWriteLock进行读写锁的操作

```
public class Main {

    ReentrantReadWriteLock lock = new ReentrantReadWriteLock();

    public static void main(String c[]) {

        // 1、一定要这样才能保证使用的是一个Lock对象，否则锁会看起来失效
        Main main = new Main();
        // 2、Thread A 【读取】
        Thread threada = new Thread(new Runnable() {
            @Override
            public void run() {
                main.read("thread A");
            }
        });
        threada.start();
        // 3、Thread B 【读取】
        Thread threadb = new Thread(new Runnable() {
            @Override
            public void run() {
                main.read("thread B");
            }
        });
        threadb.start();
        // 4、Thread C 【写入】
        Thread threadc = new Thread(new Runnable() {
            @Override
            public void run() {
                main.write("thread C");
            }
        });
        threadc.start();

    }
    // 获取【读锁】然后Read操作
    public void read(String threadName){
        lock.readLock().lock();
        try {
            long start = System.currentTimeMillis();

            while(System.currentTimeMillis() - start <= 1) {
                System.out.println(threadName+" 正在进行读操作");
            }
            System.out.println(threadName+" 读操作完毕");
        }finally {
            lock.readLock().unlock();
        }

    }
    // 获取【写锁】然后Write操作
    public void write(String threadName){
        lock.writeLock().lock();
        try {
            long start = System.currentTimeMillis();
```

```
            while(System.currentTimeMillis() - start <= 1) {
                System.out.println(threadName+" 正在进行写操作");
            }
            System.out.println(threadName+" 写操作完毕");
        }finally {
            lock.writeLock().unlock();
        }
    }
}
```

> 1、注意点：如果Lock锁失效，一定要检查不同线程中是否用的是同一个Lock！！！

74、现在有T1、T2、T3三个线程，你怎样保证T2在T1执行完后执行，T3在T2执行完后执行?

> 考察对Join是否熟悉：父线程会等待子线程运行结束

```
Thread thread3 = new Thread(new Runnable() {
            @Override
            public void run() {
              /**
                * T2：T3等待T2执行完毕
                */
                Thread thread2 = new Thread(new Runnable() {
                    @Override
                    public void run() {
                       /**
                        * T1：T2等待T1执行完毕
                        */
                        Thread thread1 = new Thread(new Runnable() {
                            @Override
                            public void run() {
                                System.out.println("Thread 1 stopped");
                            }
                        });
                        thread1.start();
                        // T2等待T1执行完毕
                        thread1.join();

                        System.out.println("Thread 2 stopped");
                    }
                });
                thread2.start();
                //T3等待T1执行完毕
                thread2.join();

                System.out.println("Thread 3 stopped");
            }
});
thread3.start();
```

锁的概念

可重入锁

1、什么是可重入锁?

> 1. 如果锁具备可重入性，则称作可重入锁。
> 2. synchronized和ReentrantLock都是可重入锁。
> 3. 线程执行到synchronized方法method1，内部调用synchronized方法method2，但是此时线程不必重新去申请锁。

```
class MyClass {
    public synchronized void method1() {
        method2();
    }

    public synchronized void method2() {
    }
}
```

> 1. 如果不具备可重入性，会导致，线程调用method1()时持有锁，但是在method2()时需要去申请锁，造成死锁。
> 2. synchronized和Lock都具有可重入性。

可中断锁

1、什么是可中断锁

> 1. 可以响应中断的锁
> 2. synchronized不是可中断锁；Lock是可中断锁。
> 3. 线程在等待锁的时候，可以自己中断自己或者在别的线程中中断自己。

公平锁

1、什么是公平锁?

> 公平锁：尽量以请求锁的顺序来获取锁。
> synchronized是非公平锁，无法保证获取锁的顺序。
> ReentrantLock和ReentrantReadWriteLock默认是非公平锁。但是可以设置为公平锁。

2、ReentrantLock如何设置为公平锁?

> 1. 通过构造方法： `ReentrantLock lock = new ReentrantLock(true);`
> 2. ReentrantLock提供了公平锁的相关方法和其他API

```
//判断锁是否是【公平锁】
   isFair()
//判断锁是否被任何线程获取
   isLocked()
 //判断锁是否被当前线程获取
   isHeldByCurrentThread()
//判断是否有线程在等待该锁
   hasQueuedThreads()
```

3、ReentrantReadWriteLock如何设置为公平锁?

> 1. 通过构造方法： `ReentrantReadWriteLock lock = new ReentrantReadWriteLock(true);`

读写锁

1、什么是读写锁?

> 1. 读写锁将对一个资源的访问，分为读锁和写锁。
> 2. ReentrantReadWriteLock就是读写锁，实现了ReadWriteLock接口。

面试题

1、在Java中Lock接口比synchronized块的优势是什么？你需要实现一个高效的缓存，它允许多个用户读，但只允许一个用户写，以此来保持它的完整性，你会怎样去实现它?

> 1. Lock可以知道线程是否获得了锁
> 2. Lock可以中断正在等待锁的线程(可中断锁)
> 3. Lock可以支持多线程读写问题(可读写锁)
> 4. Lock可以支持公平锁(通过构造方法生成)
> 5. Lock根据读写锁的特性，在资源激烈竞争的情况下，性能更好。

2、lock实际应用

> 1. lock接口在多线程和并发编程中最大的优势是它们为读和写分别提供了锁
> 2. 能满足ConcurrentHashMap这样的高性能数据结构和有条件的阻塞。

3、在java中wait和sleep方法的不同和联系?

> 1. wait在进入等待前会释放锁；sleep在等待时会一直持有锁。
> 2. wait多用于线程间交互；sleep多用于暂停执行。
> 3. wait和sleep都可以用于从线程 可运行状态 进入 超时等待状态

4、用Java实现阻塞队列。

> 1. wait()和notify()方法来实现
> 2. Lock来实现

5、用Java写代码来解决生产者——消费者问题。

> 1. wait()和notify()方法来实现
> 2. 阻塞队列

6、用Java解决哲学家进餐问题

7、用Java编程一个会导致死锁的程序，你将怎么解决?

> 1. 2个线程2个资源，竞争导致死锁。

```
public class Main {

    Object o1 = new Object();
    Object o2 = new Object();

    public static void main(String c[]) {

        Main main = new Main();

        Thread threadA = new Thread(new Runnable() {
            @Override
            public void run() {
                main.read1("thread A");
            }
        });
        threadA.start();

        Thread threadB = new Thread(new Runnable() {
            @Override
            public void run() {
                main.read2("thread B");
            }
        });
        threadB.start();

    }
    public void read1(String threadName){
        synchronized (o1){
            System.out.println(threadName + "获取到资源1，开始执行。");
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            synchronized (o2){
                System.out.println(threadName + "获取到资源2，继续执行。");
            }
            System.out.println(threadName + "执行结束。");
        }
    }

    public void read2(String threadName){
        synchronized (o2){
            System.out.println(threadName + "获取到资源2，开始执行。");
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
            synchronized (o1){
                System.out.println(threadName + "获取到资源1，继续执行。");
            }
            System.out.println(threadName + "执行结束。");
        }
```

```
    }

}
```

8、实现线程A和线程B打印1、2、3，在线程A打印好1后，线程B开始打印1、2、3，最后线程A打印2、3。

```
public class Main {

    static Lock mLock = new ReentrantLock();
    // 1、如果线程A没有打印出1，线程B就一直去等待条件满足(firstThreadIsRunning = true)
    static boolean firstThreadIsRunning = false;
    static Condition firstRuncondition = mLock.newCondition();
    // 2、线程A去等待线程B打印好1、2、3后，继续打印1、2
    static Condition waitSecondCondition = mLock.newCondition();

    public static void main(String c[]) {
        Thread thread1 = new Thread(new Runnable1());
        thread1.start();
        Thread thread2 = new Thread(new Runnable2());
        thread2.start();
    }

    static class Runnable1 implements Runnable{
        @Override
        public void run() {
            mLock.lock();
            try {
                for (int i = 0; i < 3; i++) {
                    if(i == 1){
                        firstThreadIsRunning = true;
                        // 更改flag后，唤醒第二个线程开始执行
                        firstRuncondition.signal();
                        // 等待第二个线程执行完毕(打印1、2、3)
                        waitSecondCondition.await();
                    }
                    System.out.println("A"+(i+1));
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            } finally {
                mLock.unlock();
            }
        }
    }

    static class Runnable2 implements Runnable{
        @Override
        public void run() {
            mLock.lock();
            try {
                // 必须要等到第一个Thread执行OK，才会继续执行
                while(firstThreadIsRunning == false){
                    firstRuncondition.await();
                }
                for (int i = 0; i < 3; i++) {
                    System.out.println("B"+(i+1));
                }
                waitSecondCondition.signal();
            } catch (InterruptedException e) {
                e.printStackTrace();
```

```
            }finally {
                mLock.unlock();
            }
        }
    }
}
```

8、什么是原子操作，Java中的原子操作是什么?

非常简单的java线程面试问题，接下来的问题是你需要同步一个原子操作。

9、Java中的volatile关键是什么作用？怎样使用它？在Java中它跟synchronized方法有什么不同?

自从Java 5和Java内存模型改变以后，基于volatile关键字的线程问题越来越流行。应该准备好回答关于volatile变量怎样在并发环境中确保可见性、顺序性和一致性。

10、什么是竞争条件？你怎样发现和解决竞争?

这是一道出现在多线程面试的高级阶段的问题。大多数的面试官会问最近你遇到的竞争条件，以及你是怎么解决的。有些时间他们会写简单的代码，然后让你检测出代码的竞争条件。可以参考我之前发布的关于Java竞争条件的文章。在我看来这是最好的java线程面试问题之一，它可以确切的检测候选者解决竞争条件的经验，or writing code which is free of data race or any other race condition。关于这方面最好的书是《Concurrency practices in Java》。

11、你将如何使用thread dump？你将如何分析Thread dump?

在UNIX中你可以使用kill -3，然后thread dump将会打印日志，在windows中你可以使用”CTRL+Break”。非常简单和专业的线程面试问题，但是如果他问你怎样分析它，就会很棘手。

12、为什么我们调用start()方法时会执行run()方法，为什么我们不能直接调用run()方法?

这是另一个非常经典的java多线程面试问题。这也是我刚开始写线程程序时候的困惑。现在这个问题通常在电话面试或者是在初中级Java面试的第一轮被问到。这个问题的回答应该是这样的，当你调用start()方法时你将创建新的线程，并且执行在run()方法里的代码。但是如果你直接调用run()方法，它不会创建新的线程也不会执行调用线程的代码。阅读我之前写的《start与run方法的区别》这篇文章来获得更多信息。

13、Java中你怎样唤醒一个阻塞的线程?

这是个关于线程和阻塞的棘手的问题，它有很多解决方法。如果线程遇到了IO阻塞，我并且不认为有一种方法可以中止线程。如果线程因为调用wait()、sleep()、或者join()方法而导致的阻塞，你可以中断线程，并且通过抛出InterruptedException来唤醒它。我之前写的《How to deal with blocking methods in java》有很多关于处理线程阻塞的信息。

14、在Java中CycliBarriar和CountdownLatch有什么区别?

这个线程问题主要用来检测你是否熟悉JDK5中的并发包。这两个的区别是CyclicBarrier可以重复使用已经通过的障碍，而CountdownLatch不能重复使用。

15、什么是不可变对象，它对写并发应用有什么帮助?

另一个多线程经典面试问题，并不直接跟线程有关，但间接帮助很多。这个java面试问题可以变的非常棘手，如果他要求你写一个不可变对象，或者问你为什么String是不可变的。

16、你在多线程环境中遇到的共同的问题是什么？你是怎么解决它的?

多线程和并发程序中常遇到的有Memory-interface、竞争条件、死锁、活锁和饥饿。问题是没有止境的，如果你弄错了，将很难发现和调试。这是大多数基于面试的，而不是基于实际应用的Java线程问题。

补充的其它几个问题：

1. 在java中绿色线程和本地线程区别?
2. 线程与进程的区别?
3. 什么是多线程中的上下文切换?

4)死锁与活锁的区别，死锁与馅饼的区别?

5. Java中用到的线程调度算法是什么?
6. 在Java中什么是线程调度?
7. 在线程中你怎么处理不可捕捉异常?
8. 什么是线程组，为什么在Java中不推荐使用?
9. 为什么使用Executor框架比使用应用创建和管理线程好?

> 1. 减少线程创建和销毁的开销。
> 2. Executor将任务的提交和任务的处理相分离，便于管理线程。

10. 在Java中Executor和Executors的区别?

> 1. Executor是Java线程池的顶级接口，ExecutorService直接进程子ExecutorService
> 2. Executors是一个类, 提供了若干个静态方法，用于生成不同类型的线程池。

11. 如何在Windows和Linux上查找哪个线程使用的CPU时间最长?

参考资料

1. Java并发编程：Lock
2. JAVA Lock接口详解
3. java.util.concurrent 包中Executor与Executors的区别
4. 死磕Java并发
5. Java多线程面试题Top50