0. 前言

在任何 **Java** 面试当中多线程和并发方面的问题都是必不可少的一部分，本文汇总了常见的一些多线程面试题。

一些问题，比如 volatile 关键词的作用，synchronized 和 ReentrantLock 的区别，wait()和 sleep()的区别等等问题，已经在之前写过的文章中提到过了，这里就不赘述了，有兴趣可以查看以下几篇文章：Java 并发——线程同步 volatile 与 synchronized 详解、Java 技术——Java 多线程学习、Java 并发——synchronized 和 ReentrantLock 的联系与区别。

下面是总结的之前没有提到过的面试重点题。转载请注明出处：

http://blog.csdn.net/seu_calvin/article/details/52411531

1. 多线程有什么用

（1）发挥多核 CPU 的优势

如果是单线程的程序，那么在双核 CPU 上就浪费了 50%，在 4 核 CPU 上就浪费了 75%。多线程可以真正发挥出多核 CPU 的优势来，达到充分利用 CPU 的目的。

（2）防止阻塞

多条线程同时运行，一条线程的代码执行阻塞，也不会影响其它任务的执行。

2. Runnable 接口和 Callable 接口的区别

Runnable 接口中的 run()方法的返回值是 void，它只是纯粹地去执行 run()方法中的代码而已；

Callable 接口中的 call()方法是有返回值的，是一个泛型，和 Future、FutureTask 配合可以用来获取异步执行的结果。

3. CyclicBarrier 和 CountDownLatch 的区别

两个类都在 java.util.concurrent 下，都可以用来表示代码运行到某个点上，二者的区别在于：

（1）CyclicBarrier 的某个线程运行到某个点上之后，该线程即停止运行，直到所有的线程都到达了这个点，所有线程才重新运行；CountDownLatch 则不是，某线程运行到某个点上之后，该线程会继续运行。

（2）CyclicBarrier 只能唤起一个任务，CountDownLatch 可以唤起多个任务。

（3）CyclicBarrier 可重用，CountDownLatch 不可重用，计数值为 0 该 CountDownLatch 就不可再用了

4. 线程安全的级别

代码在多线程下执行和在单线程下执行永远都能获得一样的结果，那么代码就是线程安全的。

线程安全也是有级别之分的：

（1）不可变

像 String、Integer、Long 这些，都是 final 类型的类，任何一个线程都改变不了它们的值，要改变除非新创建一个，因此这些不可变对象不需要任何同步手段就可以直接在多线程环境下使用

（2）绝对线程安全

不管运行时环境如何，调用者都不需要额外的同步措施。Java 中有绝对线程安全的类，比如 CopyOnWriteArrayList、CopyOnWriteArraySet。

（3）相对线程安全

相对线程安全也就是我们通常意义上所说的线程安全，像 Vector 这种，add、remove 方法都是原子操作，不会被打断，但也仅限于此，如果有个线程在遍历某个 Vector，同时另一个线程在 add 这个 Vector，99%的情况下都会出现 ConcurrentModificationException，也就是 fail-fast 机制。

（4）线程非安全

这个就没什么好说的了，ArrayList、LinkedList、HashMap 等都是线程非安全的类

5. 如何在两个线程之间共享数据

通过在线程之间共享对象就可以了，然后通过 wait/notify/notifyAll、await/signal/signalAll 进行唤起和等待，比方说阻塞队列 BlockingQueue 就是为线程之间共享数据而设计的

6. 为什么 wait()方法和 notify()/notifyAll()方法要在同步块中被调用

这是 JDK 强制的，wait()方法和 notify()/notifyAll()方法（都是 Object 的方法）在调用前都必须先获得对象的锁。

7. wait()方法和 notify()/notifyAll()方法在放弃对象监视器时有什么区别

wait()方法立即释放对象监视器，notify()/notifyAll()方法则会等待线程剩余代码执行完毕才会放弃对象监视器。

8. 怎么检测一个线程是否持有对象监视器

Thread 类提供了一个 holdsLock(Object obj)方法，当且仅当对象 obj 的监视器被某条线程持有的时候才会返回 true，注意这是一个 static 方法，这意味着“某条线程”指的是当前线程。

9. ConcurrentHashMap 的并发度是什么

ConcurrentHashMap 的并发度就是 segment 的大小，默认为 16，这意味着最多可以同时有 16 条线程操作 ConcurrentHashMap，这也是 ConcurrentHashMap 对 Hashtable 的最大优势。

10. ReadWriteLock 是什么

不是说 ReentrantLock 不好，只是 ReentrantLock 某些时候有所局限。

如果使用 ReentrantLock 是为了防止线程 A 在写数据、线程 B 在读数据造成的数据不一致（读和写同时操作造成的）。那么如果线程 C 在读数据，线程 D 也在读数据，读数据是不会改变数据的，那就没有必要加锁，但是还是 ReentrantLock 还是加锁了，这很显然降低了程序的性能。

读写锁 ReadWriteLock 应运而生。ReadWriteLock 是一个读写锁接口，ReentrantReadWriteLock 是 ReadWriteLock 接口的一个具体实现，实现了读写的分离：读锁是共享的，写锁是独占的，读和读之间不会互斥，读和写、写和读、写和写之间才会互斥，提升了读写的性能。

11. 如果你提交任务时，线程池队列已满，这时会发生什么

如果你使用的 LinkedBlockingQueue，也就是无界队列的话，没关系，继续添加任务到阻塞队列中等待执行，因为 LinkedBlockingQueue 可以近乎认为是一个无穷大的队列，可以无限存放任务；

如果你使用的是有界队列比方说 ArrayBlockingQueue 的话，任务首先会被添加到 ArrayBlockingQueue 中，ArrayBlockingQueue 满了，则会使用拒绝策略。

12. Java 中用到的线程调度算法是什么

抢占式：一个线程用完 CPU 之后，操作系统会根据线程优先级、线程饥饿情况等数据算出一个总的优先级并分配下一个时间片给某个线程执行。为了让某些低优先级的线程也能获取到 CPU 控制权，平衡 CPU 控制权，可以使用 Thread.sleep(0)手动触发一次操作系统分配时间片的操作。

13. 多线程中的忙循环是什么

忙循环就是程序员用循环让一个线程等待，不像传统方法 wait()，sleep() 或 yield() 它们都放弃了 CPU 控制，而忙循环不会放弃 CPU，它就是在运行一个空循环。这么做的目的是为了保留 CPU 缓存（在多核系统中，一个等待线程醒来的时候可能会在另一个内核运行，这样会重建缓存）。为了避免重建缓存和减少等待重建的时间就可以使用它了。

14. 什么是自旋

很多 synchronized 里面的代码只是一些很简单的代码，执行时间非常快，此时等待的线程都加锁可能是一种不太值得的操作，不妨让等待锁的线程不要被阻塞，而是在 synchronized 的边界做忙循环，这就是自旋。如果做了多次忙循环发现还没有获得锁，再阻塞，这样可能是一种更好的策略。

15. 什么是乐观锁和悲观锁

（1）乐观锁：乐观锁认为竞争不总是会发生，因此它不需要持有锁，将比较-设置这两个动作作为一个原子操作尝试去修改内存中的变量，如果失败则表示发生冲突，继而执行相应的重试逻辑。

（2）悲观锁：悲观锁认为竞争总是会发生，因此每次对某资源进行操作时，都会持有一个独占的锁，就像 synchronized。

16. 实现一个死锁

当线程需要同时持有多个锁时，有可能产生死锁。考虑如下情形：

（1）线程 A 当前持有互斥所锁 lock1，线程 B 当前持有互斥锁 lock2。

（2）线程 A 试图获取 lock2，因为线程 B 正持有 lock2，因此线程 A 会阻塞等待线程 B 对 lock2 释放。

（3）如果此时线程 B 也在试图获取 lock1，同理线程也会阻塞。

（4）两者都在等待对方所持有但是双方都不释放的锁，这时便会一直阻塞形成死锁。

[java] view plain copy

```java
//存放两个资源等待被使用
public class Resource {
public static Object obj1 = new Object();
    public static Object obj2 = new Object();
}
//线程 1
public class DeadThread1 implements Runnable {
    @Override
    public void run() {
    synchronized (Resource.obj1) {
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {}
    synchronized (Resource.obj2) {
        System.out.println("DeadThread1 ");
       }
}
}
}
//线程 2
public class DeadThread2 implements Runnable {
    @Override
    public void run() {
    synchronized (Resource.obj2) {
    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {}
    synchronized (Resource.obj1) {
        System.out.println("DeadThread2 ");
       }
}
}
}

//主函数中调用
Thread t1 = new Thread(new DeadThread1());
Thread t2 = new Thread(new DeadThread2());
//启动两个线程
t1.start();
t2.start();
```

17. 线程类的构造方法、静态块是被哪个线程调用的

被 new 这个线程类所在的线程所调用的，而 run 方法里面的代码才是被线程自身所调用的。

18. 锁粗化是什么意思

同步块之外的代码是异步执行的，这比同步整个方法更提升代码的效率。因此也很容易理解同步的范围越少越好的意义。但是 Java 虚拟机中存在着一种叫做锁粗化的优化方法，这种方法就是把同步范围变大。

比方说 StringBuffer，它是一个线程安全的类，反复 append 字符串意味着要进行反复的加解锁，这对性能不利，因为 JVM 在这条线程上要反复地在内核态和用户态之间切换，因此 JVM 会将多次 append 方法调用的代码进行一个锁粗的操作，将多次的 append 的操作扩展到 append 方法的头尾，变成一个大的同步块，从而提升代码执行效率。