java-大数据基础面试思考

Java集合类：

1.Java的HashMap是如何工作的？

HashMap是一个针对数据结构的键值，每个键都会有相应的值，关键是识别这样的值。

HashMap 基于 hashing 原理，我们通过 put ()和 get ()方法储存和获取对象。当我们将键值对传递给 put ()方法时，它调用键对象的 hashCode ()方法来计算 hashcode，让后找到 bucket 位置来储存值对象。当获取对象时，通过键对象的 equals ()方法找到正确的键值对，然后返回值对象。HashMap 使用 LinkedList 来解决碰撞问题，当发生碰撞了，对象将会储存在 LinkedList 的下一个节点中。 HashMap 在每个 LinkedList 节点中储存键值对对象。

2.什么是快速失败的故障安全迭代器？

快速失败的Java迭代器可能会引发ConcurrentModifcationException在底层集合迭代过程中被修改。故障安全作为发生在实例中的一个副本迭代是不会抛出任何异常的。快速失败的故障安全范例定义了当遭遇故障时系统是如何反应的。例如，用于失败的快速迭代器ArrayList和用于故障安全的迭代器ConcurrentHashMap。

3.Java BlockingQueue是什么？

Java BlockingQueue是一个并发集合util包的一部分。BlockingQueue队列是一种支持操作，它等待元素变得可用时来检索，同样等待空间可用时来存储元素。

4.什么时候使用ConcurrentHashMap？

在问题2中我们看到ConcurrentHashMap被作为故障安全迭代器的一个实例，它允许完整的并发检索和更新。当有大量的并发更新时，ConcurrentHashMap此时可以被使用。这非常类似于Hashtable，但ConcurrentHashMap不锁定整个表来提供并发，所以从这点上ConcurrentHashMap的性能似乎更好一些。所以当有大量更新时ConcurrentHashMap应该被使用。

5.哪一个List实现了最快插入？

LinkedList和ArrayList是另个不同变量列表的实现。ArrayList的优势在于动态的增长数组，非常适合初始时总长度未知的情况下使用。LinkedList的优势在于在中间位置插入和删除操作，速度是最快的。

LinkedList实现了List接口，允许null元素。此外LinkedList提供额外的get，remove，insert方法在LinkedList的首部或尾部。这些操作使LinkedList可被用作堆栈（stack），队列（queue）或双向队列（deque）。

ArrayList实现了可变大小的数组。它允许所有元素，包括null。 每个ArrayList实例都有一个容量（Capacity），即用于存储元素的数组的大小。这个容量可随着不断添加新元素而自动增加，但是增长算法并没有定义。当需要插入大量元素时，在插入前可以调用ensureCapacity方法来增加ArrayList的容量以提高插入效率。

6.Iterator和ListIterator的区别

●ListIterator有add()方法，可以向List中添加对象，而Iterator不能。
●ListIterator和Iterator都有hasNext()和next()方法，可以实现顺序向后遍历，但是ListIterator有hasPrevious()和previous()方法，可以实现逆向（顺序向前）遍历。Iterator就不可以。

●ListIterator可以定位当前的索引位置，nextIndex()和previousIndex()可以实现。Iterator没有此功能。
●都可实现删除对象，但是ListIterator可以实现对象的修改，set()方法可以实现。Iierator仅能遍历，不能修改。

7.什么是CopyOnWriteArrayList，它与ArrayList有何不同？

CopyOnWriteArrayList是ArrayList的一个线程安全的变体，其中所有可变操作（add、set等等）都是通过对底层数组进行一次新的复制来实现的。相比较于ArrayList它的写操作要慢一些，因为它需要实例的快照。

CopyOnWriteArrayList中写操作需要大面积复制数组，所以性能肯定很差，但是读操作因为操作的对象和写操作不是同一个对象，读之间也不需要加锁，读和写之间的同步处理只是在写完后通过一个简单的"="将引用指向新的数组对象上来，这个几乎不需要时间，这样读操作就很快很安全，适合在多线程里使用，绝对不会发生ConcurrentModificationException，因此CopyOnWriteArrayList适合使用在读操作远远大于写操作的场景里，比如缓存。

8.迭代器和枚举之间的区别

如果面试官问这个问题，那么他的意图一定是让你区分Iterator不同于Enumeration的两个方面：

●Iterator允许移除从底层集合的元素。
●Iterator的方法名是标准化的。

9.Hashmap如何同步?

当我们需要一个同步的HashMap时，有两种选择：

●使用Collections.synchronizedMap（..）来同步HashMap。
●使用ConcurrentHashMap的

这两个选项之间的首选是使用ConcurrentHashMap，这是因为我们不需要锁定整个对象，以及通过ConcurrentHashMap分区地图来获得锁。

10.IdentityHashMap和HashMap的区别

IdentityHashMap是Map接口的实现。不同于HashMap的，这里采用参考平等。

●在HashMap中如果两个元素是相等的，则key1.equals(key2)

●在IdentityHashMap中如果两个元素是相等的，则key1 == key2

Java多线程：
1、进程与线程区别以及线程相关概念
进程就是运行中的程序，每个进程占用独自的内存空间；线程属于进程，一个进程可以有一个或多个线程，这些线程共享这个进程的内存或系统资源，线程的切换比进程切换的负担要小。一个Java应用总是从main()方法开始运行，mian()方法运行在一个线程内，它被称为主线程。多线程的最终目的是尽可能的利用cpu资源，不让其闲置。

Thread
基础知识

2、两种创建线程的方式

（1）继承Thread类，实现其run方法.

```
Class Thread1 extends Thread{

public void run(){执行代码};

}
```


（2）实现Runnale接口，实现run方法.

```
Class Thread2 implements Runnable{

public void run(){执行代码};

}
```


3、Thread源代码分析特点

（1）Thread类实现了Runnale接口，因为实现了其run方法

（2）当生成一个线程对象的时候，如果没有为其设定名字，线程对象将使用如下形式：Thread-number(该number是自动增加的并共享于该类其它对象)

（3）两种方法均需要执行start方法分配必须的系统资源，调度线程运行并执行run

（4）在具体应用中，采用那种方式看情况而定，但当一个线程继承了另外一个类时，只能实现Runnable接口

4、线程的生命周期

概念：一个线程从创建到消亡的过程。

状态：

（1）创建状态：new---start之间称为创建状态，创建状态的线程是一个空线程系统不为其非配资源。

（2）可运行状态：start--run可运行但并不是一定在运行，只是拥有了运行的条件。

（3）不可运行状态：当调用了sleep方法或者wait方法。在指定时间后恢复或调用notify相关方法恢复。

（4）退出状态：调用了stop方法或者自然消亡。线程的停止:线程的消亡不能通过一个stop()命令，而是让run方法自然结束。可以在while循环里面条件判断break或者return。

5、线程的优先级

1.线程的优先级及其设置

目的:设置优先级是为了在多线程环境中便于系统对线程的调度，优先级高的线程将优先执行。

原则：

----线程创建时，子继承父的优先级

----setPriority()方法改变优先级

----优先级数由低到高是1——10的正整数，默认为5.（动态）

2.线程的调度策略

线程调度器选择优先级最高的线程运行，但是，如果发生以下情况，就会终止线程的运行：

（1）线程体中调用了yield方法让出了对cpu的占用权利

（2）线程体中调用了sleep方法使线程进入睡眠状态

（3）线程由于IO操作受到阻塞

（4）另外一个更高优先级线程出现

（5）在支持时间片的系统中，该线程的时间片用完。

6、关于成员变量和局部变量

如果一个变量是成员变量，那么多个线程对同一个对象的成员变量进行操作的时候，他们对该成员变量是彼此影响的，也就是说一个线程对成员变量的改变会影响到另外一个线程；如果一个变量是局部变量，那么每个线程都会有一个该局部变量的拷贝，一个线程对该局部变量的改变不会影响到其它的线程。

7、多线程同步问题（重点）

为什么要引入同步机制？在多线程环境中，可能会有两个甚至更多的线程试图同时访问一个有限的资源。必须对这种潜在资源冲突进行预防。

解决方法：在线程使用一个资源时为其加锁即可。访问资源的第一个线程为其加上锁以后，其它线程便不能在使用那个资源，除非被解决。

Synchronized：

当Synchronized关键字修饰一个方法的时候，该方法叫做同步方法：java中的每个对象都有一个锁（lock）或者叫做监视器（monitor）,当访问某个对象的synchronized方法的时候，表示将对象上锁，此时其它任何线程都无法再去访问synchronized方法了，直到之前的那个线程执行方法完毕后（或者是抛出了异常），那么将该对象的锁释放掉，其他线程才有可能再去访问该synchronized方法。

注意1：

如果一个对象有多个synchronized方法，某一个时刻某个线程已经进入到了某个synchronized方法，那么在该方法没有执行完毕前，其它线程是无法访问该对象的任何synchronzed方法的。

注意2：

如果某个Synchronized方法是static的，那么当线程访问该方法时，它锁的并不是Synchronized方法所在的对象，而是Synchronized方法所在的对象所对象的Class对象，因为java中无论一个类有多少个对象，这些对象会对应唯一一个class对象，因此当线程分别访问同一个类的两个对象的两个static Synchronized方法的时候，他们执行的顺序也是顺序的，也就是说一个线程先去执行方法，执行完毕后另一个线程才开始执行。

```
synchronized块

写法：synchronized（object）{

    方法体...

}

表示线程在执行的时候会对object对象上锁
```


synchronized方法和块比较

synchronized方法是一种粗粒度的并发控制，某一个时刻，只能有一个线程执行该synchronized方法，而synchronized快则是一种细粒度的并发控制，只会将快种的代码同步，位于方法内，synchronized块外之外的代码是可以被多个线程同时访问到的。

9、wait与notify方法

wait与notify方法都是定义在Object类中，而且是final的，因此会被所有的java类所继承并且无法重写，这两个方法要求在调用时线程应该已经获得了对象的锁，因此对这两个方法的调用需要方法synchronized方法或者块中，当线程执行了wait方法时，它会释放掉对象的锁。

10、sleep

另外一个会导致线程暂停的方法就是Thread类的sleep方法，它会导致线程睡眠指定的毫秒数，但线程在睡眠的过程中是不会释放掉对象的锁的。

50道Java线程面试题

下面是Java线程相关的热门面试题，你可以用它来好好准备面试。

1) 什么是线程？

线程是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位。程序员可以通过它进行多处理器编程，你可以使用多线程对运算密集型任务提速。比如，如果一个线程完成一个任务要100毫秒，那么用十个线程完成改任务只需10毫秒。Java在语言层面对多线程提供了卓越的支持，它也是一个很好的卖点。欲了解更多详细信息请点击这里。

2) 线程和进程有什么区别？

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。别把它和栈内存搞混，每个线程都拥有单独的栈内存用来存储本地数据。更多详细信息请点击这里。

3) 如何在Java中实现线程？

在语言层面有两种方式。java.lang.Thread 类的实例就是一个线程但是它需要调用java.lang.Runnable接口来执行，由于线程类本身就是调用的Runnable接口所以你可以继承java.lang.Thread 类或者直接调用Runnable接口来重写run()方法实现线程。更多详细信息请点击这里.

4) 用Runnable还是Thread?

这个问题是上题的后续，大家都知道我们可以通过继承Thread类或者调用Runnable接口来实现线程，问题是，那个方法更好呢？什么情况下使用它？这个问题很容易回答，如果你知道Java不支持类的多重继承，但允许你调用多个接口。所以如果你要继承其他类，当然是调用Runnable接口好了。更多详细信息请点击这里。

6) Thread 类中的start() 和 run() 方法有什么区别？

这个问题经常被问到，但还是能从此区分出面试者对Java线程模型的理解程度。start()方法被用来启动新创建的线程，而且start()内部调用了run()方法，这和直接调用run()方法的效果不一样。当你调用run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。更多讨论请点击这里

7) Java中Runnable和Callable有什么不同？

Runnable和Callable都代表那些要在不同的线程中执行的任务。Runnable从JDK1.0开始就有了，Callable是在JDK1.5增加的。它们的主要区别是Callable的 call() 方法可以返回值和抛出异常，而Runnable的run()方法没有这些功能。Callable可以返回装载有计算结果的Future对象。我的博客有更详细的说明。

8) Java中CyclicBarrier 和 CountDownLatch有什么不同？

CyclicBarrier 和 CountDownLatch 都可以用来让一组线程等待其它线程。与 CyclicBarrier 不同的是，CountdownLatch 不能重新使用。点此查看更多信息和示例代码。

9) Java内存模型是什么？

Java内存模型规定和指引Java程序在不同的内存架构、CPU和操作系统间有确定性地行为。它在多线程的情况下尤其重要。Java内存模型对一个线程所做的变动能被其它线程可见提供了保证，它们之间是先行发生关系。这个关系定义了一些规则让程序员在并发编程时思路更清晰。比如，先行发生关系确保了：

- 线程内的代码能够按先后顺序执行，这被称为程序次序规则。
- 对于同一个锁，一个解锁操作一定要发生在时间上后发生的另一个锁定操作之前，也叫做管程锁定规则。
- 前一个对volatile的写操作在后一个volatile的读操作之前，也叫volatile变量规则。
- 一个线程内的任何操作必需在这个线程的start()调用之后，也叫作线程启动规则。
- 一个线程的所有操作都会在线程终止之前，线程终止规则。
- 一个对象的终结操作必需在这个对象构造完成之后，也叫对象终结规则。
- 可传递性

我强烈建议大家阅读《Java并发编程实践》第十六章来加深对Java内存模型的理解。

10) Java中的volatile 变量是什么？

volatile是一个特殊的修饰符，只有成员变量才能使用它。在Java并发程序缺少同步类的情况下，多线程对成员变量的操作对其它线程是透明的。volatile变量可以保证下一个读取操作会在前一个写操作之后发生，就是上一题的volatile变量规则。点击这里查看更多volatile的相关内容。

11) 什么是线程安全？Vector是一个线程安全类吗？（详见这里）

如果你的代码所在的进程中有多个线程在同时运行，而这些线程可能会同时运行这段代码。如果每次运行结果和单线程运行的结果是一样的，而且其他的变量的值也和预期的是一样的，就是线程安全的。一个线程安全的计数器类的同一个实例对象在被多个线程使用的情况下也不会出现计算失误。很显然你可以将集合类分成两组，线程安全和非线程安全的。Vector 是用同步方法来实现线程安全的, 而和它相似的ArrayList不是线程安全的。

12) Java中什么是竞态条件？ 举个例子说明。

竞态条件会导致程序在并发情况下出现一些bugs。多线程对一些资源的竞争的时候就会产生竞态条件，如果首先要执行的程序竞争失败排到后面执行了，那么整个程序就会出现一些不确定的bugs。这种bugs很难发现而且会重复出现，因为线程间的随机竞争。一个例子就是无序处理，详见答案。

13) Java中如何停止一个线程？

Java提供了很丰富的API但没有为停止线程提供API。JDK 1.0本来有一些像stop(), suspend() 和 resume()的控制方法但是由于潜在的死锁威胁因此在后续的JDK版本中他们被弃用了，之后Java API的设计者就没有提供一个兼容且线程安全的方法来停止一个线程。当run() 或者 call() 方法执行完的时候线程会自动结束,如果要手动结束一个线程，你可以用volatile 布尔变量来退出run()方法的循环或者是取消任务来中断线程。点击这里查看示例代码。

14) 一个线程运行时发生异常会怎样？

这是我在一次面试中遇到的一个很刁钻的Java面试题, 简单的说，如果异常没有被捕获该线程将会停止执行。Thread.UncaughtExceptionHandler是用于处理未捕获异常造成线程突然中断情况的一个内嵌接口。当一个未捕获异常将造成线程中断的时候JVM会使用Thread.getUncaughtExceptionHandler()来查询线程的UncaughtExceptionHandler并将线程和异常作为参数传递给handler的uncaughtException()方法进行处理。

15） 如何在两个线程间共享数据？

你可以通过共享对象来实现这个目的，或者是使用像阻塞队列这样并发的数据结构。这篇教程《Java线程间通信》(涉及到在两个线程间共享对象)用wait和notify方法实现了生产者消费者模型。

16) Java中notify 和 notifyAll有什么区别？

这又是一个刁钻的问题，因为多线程可以等待单监控锁，Java API 的设计人员提供了一些方法当等待条件改变的时候通知它们，但是这些方法没有完全实现。notify()方法不能唤醒某个具体的线程，所以只有一个线程在等待的时候它才有用武之地。而notifyAll()唤醒所有线程并允许他们争夺锁确保了至少有一个线程能继续运行。我的博客有更详细的资料和示例代码。

17) 为什么wait, notify 和 notifyAll这些方法不在thread类里面？

这是个设计相关的问题，它考察的是面试者对现有系统和一些普遍存在但看起来不合理的事物的看法。回答这些问题的时候，你要说明为什么把这些方法放在Object类里是有意义的，还有不把它放在Thread类里的原因。一个很明显的原因是JAVA提供的锁是对象级的而不是线程级的，每个对象都有锁，通过线程获得。如果线程需要等待某些锁那么调用对象中的wait()方法就有意义了。如果wait()方法定义在Thread类中，线程正在等待的是哪个锁就不明显了。简单的说，由于wait，notify和notifyAll都是锁级别的操作，所以把他们定义在Object类中因为锁属于对象。你也可以查看这篇文章了解更多。

18) 什么是ThreadLocal变量？

ThreadLocal是Java里一种特殊的变量。每个线程都有一个ThreadLocal就是每个线程都拥有了自己独立的一个变量，竞争条件被彻底消除了。它是为创建代价高昂的对象获取线程安全的好方法，比如你可以用ThreadLocal让SimpleDateFormat变成线程安全的，因为那个类创建代价高昂且每次调用都需要创建不同的实例所以不值得在局部范围使用它，如果为每个线程提供一个自己独有的变量拷贝，将大大提高效率。首先，通过复用减少了代价高昂的对象的创建个数。其次，你在没有使用高代价的同步或者不变性的情况下获得了线程安全。线程局部变量的另一个不错的例子是ThreadLocalRandom类，它在多线程环境中减少了创建代价高昂的Random对象的个数。查看答案了解更多。

19) 什么是FutureTask?

在Java并发程序中FutureTask表示一个可以取消的异步运算。它有启动和取消运算、查询运算是否完成和取回运算结果等方法。只有当运算完成的时候结果才能取回，如果运算尚未完成get方法将会阻塞。一个FutureTask对象可以对调用了Callable和Runnable的对象进行包装，由于FutureTask也是调用了Runnable接口所以它可以提交给Executor来执行。

20) Java中interrupted 和 isInterruptedd方法的区别？

interrupted() 和 isInterrupted()的主要区别是前者会将中断状态清除而后者不会。Java多线程的中断机制是用内部标识来实现的，调用Thread.interrupt()来中断一个线程就会设置中断标识为true。当中断线程调用静态方法Thread.interrupted()来检查中断状态时，中断状态会被清零。而非静态方法isInterrupted()用来查询其它线程的中断状态且不会改变中断状态标识。简单的说就是任何抛出InterruptedException异常的方法都会将中断状态清零。无论如何，一个线程的中断状态有有可能被其它线程调用中断来改变。

21) 为什么wait和notify方法要在同步块中调用？

主要是因为Java API强制要求这样做，如果你不这么做，你的代码会抛出IllegalMonitorStateException异常。还有一个原因是为了避免wait和notify之间产生竞态条件。

22) 为什么你应该在循环中检查等待条件？

处于等待状态的线程可能会收到错误警报和伪唤醒，如果不在循环中检查等待条件，程序就会在没有满足结束条件的情况下退出。因此，当一个等待线程醒来时，不能认为它原来的等待状态仍然是有效的，在notify()方法调用之后和等待线程醒来之前这段时间它可能会改变。这就是在循环中使用wait()方法效果更好的原因，你可以在Eclipse中创建模板调用wait和notify试一试。如果你想了解更多关于这个问题的内容，我推荐你阅读《Effective Java》这本书中的线程和同步章节。

23) Java中的同步集合与并发集合有什么区别？

同步集合与并发集合都为多线程和并发提供了合适的线程安全的集合，不过并发集合的可扩展性更高。在Java1.5之前程序员们只有同步集合来用且在多线程并发的时候会导致争用，阻碍了系统的扩展性。Java5介绍了并发集合像ConcurrentHashMap，不仅提供线程安全还用锁分离和内部分区等现代技术提高了可扩展性。更多内容详见答案。

24） Java中堆和栈有什么不同？

为什么把这个问题归类在多线程和并发面试题里？因为栈是一块和线程紧密相关的内存区域。每个线程都有自己的栈内存，用于存储本地变量，方法参数和栈调用，一个线程中存储的变量对其它线程是不可见的。而堆是所有线程共享的一片公用内存区域。对象都在堆里创建，为了提升效率线程会从堆中弄一个缓存到自己的栈，如果多个线程使用该变量就可能引发问题，这时volatile 变量就可以发挥作用了，它要求线程从主存中读取变量的值。

更多内容详见答案。

25） 什么是线程池？ 为什么要使用它？

创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从JDK1.5开始，Java API提供了Executor框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。更多内容详见这篇文章。

26） 如何写代码来解决生产者消费者问题？

在现实中你解决的许多线程问题都属于生产者消费者模型，就是一个线程生产任务供其它线程进行消费，你必须知道怎么进行线程间通信来解决这个问题。比较低级的办法是用wait和notify来解决这个问题，比较赞的办法是用Semaphore 或者 BlockingQueue来实现生产者消费者模型，这篇教程有实现它。

27） 如何避免死锁？

Java多线程中的死锁

死锁是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。这是一个严重的问题，因为死锁会让你的程序挂起无法完成任务，死锁的发生必须满足以下四个条件：

- 互斥条件：一个资源每次只能被一个进程使用。
- 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。
- 不剥夺条件：进程已获得的资源，在末使用完之前，不能强行剥夺。
- 循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。

避免死锁最简单的方法就是阻止循环等待条件，将系统中所有的资源设置标志位、排序，规定所有的进程申请资源必须以一定的顺序（升序或降序）做操作来避免死锁。这篇教程有代码示例和避免死锁的讨论细节。

28) Java中活锁和死锁有什么区别？

这是上题的扩展，活锁和死锁类似，不同之处在于处于活锁的线程或进程的状态是不断改变的，活锁可以认为是一种特殊的饥饿。一个现实的活锁例子是两个人在狭小的走廊碰到，两个人都试着避让对方好让彼此通过，但是因为避让的方向都一样导致最后谁都不能通过走廊。简单的说就是，活锁和死锁的主要区别是前者进程的状态可以改变但是却不能继续执行。

29） 怎么检测一个线程是否拥有锁？

我一直不知道我们竟然可以检测一个线程是否拥有锁，直到我参加了一次电话面试。在java.lang.Thread中有一个方法叫holdsLock()，它返回true如果当且仅当当前线程拥有某个具体对象的锁。你可以查看这篇文章了解更多。

30) 你如何在Java中获取线程堆栈？

对于不同的操作系统，有多种方法来获得Java进程的线程堆栈。当你获取线程堆栈时，JVM会把所有线程的状态存到日志文件或者输出到控制台。在Windows你可以使用Ctrl + Break组合键来获取线程堆栈，Linux下用kill -3命令。你也可以用jstack这个工具来获取，它对线程id进行操作，你可以用jps这个工具找到id。

31) JVM中哪个参数是用来控制线程的栈堆栈小的

这个问题很简单， -Xss参数用来控制线程的堆栈大小。你可以查看JVM配置列表来了解这个参数的更多信息。

32） Java中synchronized 和 ReentrantLock 有什么不同？

Java在过去很长一段时间只能通过synchronized关键字来实现互斥，它有一些缺点。比如你不能扩展锁之外的方法或者块边界，尝试获取锁时不能中途取消等。Java 5 通过Lock接口提供了更复杂的控制来解决这些问题。 ReentrantLock 类实现了 Lock，它拥有与 synchronized 相同的并发性和内存语义且它还具有可扩展性。你可以查看这篇文章了解更多

33） 有三个线程T1，T2，T3，怎么确保它们按顺序执行？

在多线程中有多种方法让线程按特定顺序执行，你可以用线程类的join()方法在一个线程中启动另一个线程，另外一个线程完成该线程继续执行。为了确保三个线程的顺序你应该先启动最后一个(T3调用T2，T2调用T1)，这样T1就会先完成而T3最后完成。你可以查看这篇文章了解更多。

34) Thread类中的yield方法有什么作用？

Yield方法可以暂停当前正在执行的线程对象，让其它有相同优先级的线程执行。它是一个静态方法而且只保证当前线程放弃CPU占用而不能保证使其它线程一定能占用CPU，执行yield()的线程有可能在进入到暂停状态后马上又被执行。点击这里查看更多yield方法的相关内容。

35） Java中ConcurrentHashMap的并发度是什么？

ConcurrentHashMap把实际map划分成若干部分来实现它的可扩展性和线程安全。这种划分是使用并发度获得的，它是ConcurrentHashMap类构造函数的一个可选参数，默认值为16，这样在多线程情况下就能避免争用。欲了解更多并发度和内部大小调整请阅读我的文章How ConcurrentHashMap works in Java。

36） Java中Semaphore是什么？

Java中的Semaphore是一种新的同步类，它是一个计数信号。从概念上讲，从概念上讲，信号量维护了一个许可集合。如有必要，在许可可用前会阻塞每一个 acquire()，然后再获取该许可。每个 release()添加一个许可，从而可能释放一个正在阻塞的获取者。但是，不使用实际的许可对象，Semaphore只对可用许可的号码进行计数，并采取相应的行动。信号量常常用于多线程的代码中，比如数据库连接池。更多详细信息请点击这里。

37）如果你提交任务时，线程池队列已满。会时发会生什么？

这个问题问得很狡猾，许多程序员会认为该任务会阻塞直到线程池队列有空位。事实上如果一个任务不能被调度执行那么ThreadPoolExecutor's submit()方法将会抛出一个RejectedExecutionException异常。

38) Java线程池中submit() 和 execute()方法有什么区别？

两个方法都可以向线程池提交任务，execute()方法的返回类型是void，它定义在Executor接口中, 而submit()方法可以返回持有计算结果的Future对象，它定义在ExecutorService接口中，它扩展了Executor接口，其它线程池类像ThreadPoolExecutor和ScheduledThreadPoolExecutor都有这些方法。更多详细信息请点击这里。

39) 什么是阻塞式方法？

阻塞式方法是指程序会一直等待该方法完成期间不做其他事情，ServerSocket的accept()方法就是一直等待客户端连接。这里的阻塞是指调用结果返回之前，当前线程会被挂起，直到得到结果之后才会返回。此外，还有异步和非阻塞式方法在任务完成前就返回。更多详细信息请点击这里。

40) Swing是线程安全的吗？ 为什么？

你可以很肯定的给出回答，Swing不是线程安全的，但是你应该解释这么回答的原因即便面试官没有问你为什么。当我们说swing不是线程安全的常常提到它的组件，这些组件不能在多线程中进行修改，所有对GUI组件的更新都要在AWT线程中完成，而Swing提供了同步和异步两种回调方法来进行更新。点击这里查看更多swing和线程安全的相关内容。

41） Java中invokeAndWait 和 invokeLater有什么区别？

这两个方法是Swing API 提供给Java开发者用来从当前线程而不是事件派发线程更新GUI组件用的。InvokeAndWait()同步更新GUI组件，比如一个进度条，一旦进度更新了，进度条也要做出相应改变。如果进度被多个线程跟踪，那么就调用invokeAndWait()方法请求事件派发线程对组件进行相应更新。而invokeLater()方法是异步调用更新组件的。更多详细信息请点击这里。

42) Swing API中那些方法是线程安全的？

这个问题又提到了swing和线程安全，虽然组件不是线程安全的但是有一些方法是可以被多线程安全调用的，比如repaint(), revalidate()。 JTextComponent的setText()方法和JTextArea的insert() 和 append() 方法也是线程安全的。

43) 如何在Java中创建Immutable对象？

这个问题看起来和多线程没什么关系， 但不变性有助于简化已经很复杂的并发程序。Immutable对象可以在没有同步的情况下共享，降低了对该对象进行并发访问时的同步化开销。可是Java没有@Immutable这个注解符，要创建不可变类，要实现下面几个步骤：通过构造方法初始化所有成员、对变量不要提供setter方法、将所有的成员声明为私有的，这样就不允许直接访问这些成员、在getter方法中，不要直接返回对象本身，而是克隆对象，并返回对象的拷贝。我的文章how to make an object Immutable in Java有详细的教程，看完你可以充满自信。

44） Java中的ReadWriteLock是什么？

一般而言，读写锁是用来提升并发程序性能的锁分离技术的成果。Java中的ReadWriteLock是Java 5 中新增的一个接口，一个ReadWriteLock维护一对关联的锁，一个用于只读操作一个用于写。在没有写线程的情况下一个读锁可能会同时被多个读线程持有。写锁是独占的，你可以使用JDK中的ReentrantReadWriteLock来实现这个规则，它最多支持65535个写锁和65535个读锁。

45) 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。你可以查看这篇文章获得更多信息。

46） volatile 变量和 atomic 变量有什么不同？

这是个有趣的问题。首先，volatile 变量和 atomic 变量看起来很像，但功能却不一样。Volatile变量可以确保先行关系，即写操作会发生在后续的读操作之前, 但它并不能保证原子性。例如用volatile修饰count变量那么count++ 操作就不是原子性的。而AtomicInteger类提供的atomic方法可以让这种操作具有原子性如getAndIncrement()方法会原子性的进行增量操作把当前值加一，其它数据类型和引用变量也可以进行相似操作。

47) 如果同步块内的线程抛出异常会发生什么？

这个问题坑了很多Java程序员，若你能想到锁是否释放这条线索来回答还有点希望答对。无论你的同步块是正常还是异常退出的，里面的线程都会释放锁，所以对比锁接口我更喜欢同步块，因为它不用我花费精力去释放锁，该功能可以在finally block里释放锁实现。

48） 单例模式的双检锁是什么？

这个问题在Java面试中经常被问到，但是面试官对回答此问题的满意度仅为50%。一半的人写不出双检锁还有一半的人说不出它的隐患和Java1.5是如何对它修正的。它其实是一个用来创建线程安全的单例的老方法，当单例实例第一次被创建时它试图用单个锁进行性能优化，但是由于太过于复杂在JDK1.4中它是失败的，我个人也不喜欢它。无论如何，即便你也不喜欢它但是还是要了解一下，因为它经常被问到。你可以查看how double checked locking on Singleton works这篇文章获得更多信息。

49） 如何在Java中创建线程安全的Singleton？

这是上面那个问题的后续，如果你不喜欢双检锁而面试官问了创建Singleton类的替代方法，你可以利用JVM的类加载和静态变量初始化特征来创建Singleton实例，或者是利用枚举类型来创建Singleton，我很喜欢用这种方法。你可以查看这篇文章获得更多信息。

50) 写出3条你遵循的多线程最佳实践

这种问题我最喜欢了，我相信你在写并发代码来提升性能的时候也会遵循某些最佳实践。以下三条最佳实践我觉得大多数Java程序员都应该遵循：

- 给你的线程起个有意义的名字。

这样可以方便找bug或追踪。OrderProcessor, QuoteProcessor or TradeProcessor 这种名字比 Thread-1. Thread-2 and Thread-3 好多了，给线程起一个和它要完成的任务相关的名字，所有的主要框架甚至JDK都遵循这个最佳实践。

- 避免锁定和缩小同步的范围

锁花费的代价高昂且上下文切换更耗费时间空间，试试最低限度的使用同步和锁，缩小临界区。因此相对于同步方法我更喜欢同步块，它给我拥有对锁的绝对控制权。

- 多用同步类少用wait 和 notify

首先，CountDownLatch, Semaphore, CyclicBarrier 和 Exchanger 这些同步类简化了编码操作，而用wait和notify很难实现对复杂控制流的控制。其次，这些类是由最好的企业编写和维护在后续的JDK中它们还会不断优化和完善，使用这些更高等级的同步工具你的程序可以不费吹灰之力获得优化。

- 多用并发集合少用同步集合

这是另外一个容易遵循且受益巨大的最佳实践，并发集合比同步集合的可扩展性更好，所以在并发编程时使用并发集合效果更好。如果下一次你需要用到map，你应该首先想到用ConcurrentHashMap。我的文章Java并发集合有更详细的说明。

51) 如何强制启动一个线程？

这个问题就像是如何强制进行Java垃圾回收，目前还没有觉得方法，虽然你可以使用System.gc()来进行垃圾回收，但是不保证能成功。在Java里面没有办法强制启动一个线程，它是被线程调度器控制着且Java没有公布相关的API。

52) Java中的fork join框架是什么？

fork join框架是JDK7中出现的一款高效的工具，Java开发人员可以通过它充分利用现代服务器上的多处理器。它是专门为了那些可以递归划分成许多子模块设计的，目的是将所有可用的处理能力用来提升程序的性能。fork join框架一个巨大的优势是它使用了工作窃取算法，可以完成更多任务的工作线程可以从其它线程中窃取任务来执行。你可以查看这篇文章获得更多信息。

53） Java多线程中调用wait() 和 sleep()方法有什么不同？

Java程序中wait 和 sleep都会造成某种形式的暂停，它们可以满足不同的需要。wait()方法用于线程间通信，如果等待条件为真且其它线程被唤醒时它会释放锁，而sleep()方法仅仅释放CPU资源或者让当前线程停止执行一段时间，但不会释放锁。你可以查看这篇文章获得更多信息。

以上就是50道热门Java多线程和并发面试题啦。我没有分享所有题的答案但给未来的阅读者提供了足够的提示和线索来寻找答案。如果你真的找不到某题的答案，联系我吧，我会加上去的。这篇文章不仅可以用来准备面试，还能检查你对多线程、并发、设计模式和竞态条件、死锁和线程安全等线程问题的理解。我打算把这篇文章的问题弄成所有Java多线程问题的大合集，但是没有你的帮助恐怖是不能完成的，你也可以跟我分享其它任何问题，包括那些你被问到却还没有找到答案的问题。这篇文章对初学者或者是经验丰富的Java开发人员都很有用，过两三年甚至五六年你再读它也会受益匪浅。它可以扩展初学者尤其有用因为这个可以扩展他们的知识面，我会不断更新这些题，大家可以在文章后面的评论中提问，分享和回答问题一起把这篇面试题完善。