JavaSE 常见面试题-线程篇

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。

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。每个线程都拥有单独的栈内存用来存储本地数据。

3.如何在 Java 中实现线程?

一种是继承 Thread 类
另一种就是实现 Runnable 接口
最后一种就是实现 Callable 接口
(第四种也是实现 callable 接口，只不过有返回值而已)

4.Java 关键字 volatile 与 synchronized 作用与区别?

1.volatile

它所修饰的变量不保留拷贝.直接访问主内存中的。

在 Java 内存模型中.有 main memory.每个线程也有自己的 memory (例如寄存器)。为了性能.一个线程会在自己的 memory 中保持要访问的变量的副本。这样就会出现同一个变 量在某个瞬间.在一个线程的 memory 中的值可能与另外一个线程 memory 中的值.或者 main memory 中的值不一致的情况。 一个变量声明为 volatile.就意味着这个变量是随时会被其他线程修改的.因此不能将它 cache 在线程 memory 中。

2.synchronized

当它用来修饰一个方法或者一个代码块的时候.能够保证在同一时刻最多只有一个线程执行该段代码。

一、当两个并发线程访问同一个对象 object 中的这个 synchronized(this)同步代码块时.一个时间内只能有一个线程得到执行。另一个线程必须等待当前线程执行完这个代码块以后才能执行该代码块。

二、然而.当一个线程访问 object 的一个 synchronized(this)同步代码块时.另一个线程仍然可以访问该 object 中的非 synchronized(this)同步代码块。

三、尤其关键的是.当一个线程访问 object 的一个 synchronized(this)同步代码块时.其他线程对 object 中所有其它 synchronized(this)同步代码块的访问将被阻塞。

四、当一个线程访问 object 的一个 synchronized(this)同步代码块时.它就获得了这个 object 的对象锁。结果.其它线程对该 object 对象所有同步代码部分的访问都被暂时阻塞。

五、以上规则对其它对象锁同样适用.

5.有哪些不同的线程生命周期?

当我们在 Java 程序中新建一个线程时.它的状态是 *New*。当我们调用线程的 start()方法时.状态被改变为 *Runnable*。线程调度器会为 *Runnable* 线程池中的线程分配 CPU 时间并且讲它们的状态改变为 *Running*。其他的线程状态还有 *Waiting.Blocked* 和 *Dead*。

6.你对线程优先级的理解是什么?

每一个线程都是有优先级的.一般来说.高优先级的线程在运行时会具有优先权.但这依赖于线程调度的实现.这个实现是和操作系统相关的(OS dependent)。我们可以定义线程的优先级.但是这并不能保证高优先级的线程会在低优先级的线程前执行。线程优先级是一个 int 变量(从 1-10).1 代表最低优先级.10 代表最高优先级。

7.什么是死锁(Deadlock)? 如何分析和避免死锁?

死锁是指两个以上的线程永远阻塞的情况.这种情况产生至少需要两个以上的线程和两个以上的资源。

分析死锁.我们需要查看 Java 应用程序的线程转储。我们需要找出那些状态为 BLOCKED 的线程和他们等待的资源。每个资源都有一个唯一的 id.用这个 id 我们可以找出哪些线程已经拥有了它的对象锁。

避免嵌套锁.只在需要的地方使用锁和避免无限期等待是避免死锁的通常办法。

8.什么是线程安全？Vector 是一个线程安全类吗?

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。

9.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

9-1 一个线程运行时发生异常会怎样?

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。

10.什么是 ThreadLocal?

ThreadLocal 用于创建线程的本地变量.我们知道一个对象的所有线程会共享它的全局变量.所以这些变量不是线程安全的.我们可以使用同步技术。但是当我们不想使用同步的时候.我们可以选择 ThreadLocal 变量。

每个线程都会拥有他们自己的 Thread 变量.它们可以使用 get()\set()方法去获取他们的默认值或者在线程内部改变他们的值。ThreadLocal 实例通常是希望它们同线程状态关联起来是 private static 属性。

10-1ThreadLocal 可以用来共享数据吗?

不可以

ThreadLocal 是采用哈希表的方式来为每个线程都提供一个变量的副本

ThreadLocal 保证各个线程间数据安全，每个线程的数据不会被另外线程访问和破坏

1.ThreadLocal 的类声明：

public class ThreadLocal

可以看出 ThreadLocal 并没有继承自 Thread，也没有实现 Runnable 接口。所以 AB 都不对。

2.ThreadLocal 类为每一个线程都维护了自己独有的变量拷贝。每个线程都拥有了自己独立的一个变量。

所以 ThreadLocal 重要作用并不在于多线程间的数据共享，而是数据的独立，C 选项错。

由于每个线程在访问该变量时，读取和修改的，都是自己独有的那一份变量拷贝，不会被其他线程访问，

变量被彻底封闭在每个访问的线程中。所以 E 对。

3、ThreadLocal 中定义了一个哈希表用于为每个线程都提供一个变量的副本：

```
 static class ThreadLocalMap {
      static class Entry extends
WeakReference {
         /** The value associated with this ThreadLocal. */
         Object value;
         Entry(ThreadLocal k, Object v) {
            super(k);
            value = v;
         }
      }
      /**
       * The table, resized as necessary.
       * table.length MUST always be a power of two.
       */
      private Entry[] table;
}
```

11.Sleep()、suspend()和 wait()之间有什么区别?

Thread.sleep()使当前线程在指定的时间处于“非运行” （Not Runnable）状态。线程一直持有对象的监视器。比如一个线程当前在一个同步块或同步方法中.其它线程不能进入该块或方法中。如果另一线程调用了 interrupt()方法.它将唤醒那个“睡眠的”线程。

注意：sleep()是一个静态方法。这意味着只对当前线程有效.一个常见的错误是调用 t.sleep().（这里的 t 是一个不同于当前线程的线程）。即便是执行 t.sleep().也是当前线程进入睡眠.而不是 t 线程。t.suspend()是过时的方法.使用 suspend()导致线程进入停滞状态.该线程会一直持有对象的监视器.suspend()容易引起死锁问题。

object.wait()使当前线程出于“不可运行”状态.和 sleep()不同的是 wait 是 object 的方法而不是 thread。调用 object.wait()时.线程先要获取这个对象的对象锁.当前线程必须在锁对象保持同步.把当前线程添加到等待队列中.随后另一线程可以同步同一个对象锁来调用 object.notify().这样将唤醒原来等待中的线程.然后释放该锁。基本上 wait()/notify()与 sleep()/interrupt()类似.只是前者需要获取对象锁。

12.什么是线程饿死.什么是活锁?

当所有线程阻塞.或者由于需要的资源无效而不能处理.不存在非阻塞线程使资源可用。JavaAPI 中线程活锁可能发生在以下情形：

1.当所有线程在程序中执行 Object.wait(0).参数为 0 的 wait 方法。程序将发生活锁直到在相应的对象上有线程调用 Object.notify()或者 Object.notifyAll()。

2.当所有线程卡在无限循环中。

13.什么是 Java Timer 类？如何创建一个有特定时间间隔的任务?

java.util.Timer 是一个工具类.可以用于安排一个线程在未来的某个特定时间执行。Timer 类可以用安排一次性任务或者周期任务。

java.util.TimerTask 是一个实现了 Runnable 接口的抽象类.我们需要去继承这个类来创建我们自己的定时任务并使用 Timer 去安排它的执行。

14.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。

15.同步方法和同步块.哪个是更好的选择?

同步块是更好的选择.因为它不会锁住整个对象（当然你也可以让它锁住整个对象）。同步方法会锁住整个对象.哪怕这个类中有多个不相关联的同步块.这通常会导致他们停止执行并需要等待获得这个对象上的锁。

16.什么是线程池？ 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）。

17.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。

18.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。

19.多线程有几种实现方式

多线程有两种实现方法，分别是继承 Thread 类与实现 Runnable 接口

同步的实现方面有五种，分别是 synchronized、 wait 与 notify、sleep、suspend、join

synchronized: 一直持有锁，直至执行结束

wait():使一个线程处于等待状态，并且释放所持有的对象的 lock，需捕获异常。

sleep():使一个正在运行的线程处于睡眠状态，是一个静态方法，需捕获异常，不释放锁。

notify():唤醒一个处于等待状态的线程，注意的是在调用此方法的时候，并不能确切的唤醒某一个等待状态的线程，而是由 JVM 确定唤醒哪个线程，而且不是按优先级。

notityAll():唤醒所有处入等待状态的线程，注意并不是给所有唤醒线程一个对象的锁，而是让它们竞争

20.启动一个线程用 run 还是 start

启动一个线程是调用 start()方法，使线程就绪状态，以后可以被调度为运行状态，一个线程必须关联一些具体的执行代码，run()方法是该线程所关联的执行代码。

21.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

22.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。

23.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，在未使用完之前，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

24. 怎么检测一个线程是否拥有锁?

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。

25. 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，用 wait 和 notify 方法实现了生产者消费者模型。

26. 什么地方用了多线程

你可以说我们的做的项目几乎没有用到多线程

因为 java web 方面开发的话几乎用不到多线程！有多线程的地方 servlet 容器或者其他开发框架都已经实现掉了！

27.Excutors 可以产生哪些线程池

Java 里面线程池的顶级接口是 Executor，但是严格意义上讲 Executor 并不是一个线程池，而只是一个执行线程的工具。真正的线程池接口是 ExecutorService。ThreadPoolExecutor 是 Executors 类的底层实现。

28.Java 并发包当中的类，它们都有哪些作用

这些并发包类就是 java.concurrent 包下面的

比如面试官可能会先问你，如果想实现所有的线程一起等待某个事件的发生，当某个事件发生时，所有线程一起开始往下执行的话，有什么好的办法吗?

这个时候你可能会说可以用栅栏（Java 的并发包中的 CyclicBarrier）

29.Java 中多线程同步是什么?

答：即当有一个线程在对内存进行操作时，其他线程都不可以对这个内存地址进行操作，直到该线程完成操作， 其他线程才能对该内存地址进行操作，而其他线程又处于等待状态

30.线程之间如何通信?

答:多个线程在处理同一个资源，但是处理的动作（线程的任务）却不相同。通过一定的手段使各个线程能有效的利用资源。

通常情况下，一个次级线程要为主线程完成某种特定类型的任务，这就隐含着表示在主线程和次级线程之间需要建立一个通信的通道。一般情况下，有下面的几种方法实现这种通信任务：使用全局变量、使用事件对象、使用消息

31.用户线程和守护线程有什么区别

答:所谓守护线程，是指在程序运行的时候在后台提供一种通用服务的线程，比如垃圾回收线程就是一个很称职的守护者，并且这种线程并不属于程序中不可或缺的部分。因此，当所有的非守护线程结束时，程序也就终止了，同时会杀死进程中的所有守护线程。反过来说，只要任何非守护线程还在运行，程序就不会终止。

用户线程和守护线程两者几乎没有区别，唯一的不同之处就在于虚拟机的离开：如果用户线程已经全部退出运行了，只剩下守护线程存在了，虚拟机也就退出了。因为没有了被守护者，守护线程也就没有工作可做了，也就没有继续运行程序的必要了

32.如何暂停一条线程?

答：两种方式暂停一条线程，一个是采取 Thread 类的 sleep()方法,一个是在同步代码中使用 wait()方法.

33.线程的调度和时间分片?

答:java 虚拟机采用抢占式调度模型，是指优先让可运行池中优先级高的线程占用 CPU，如果可运行池中的线程优先级相同，那么就随机选择一个线程，使其占用 CPU。处于运行状态的线程会一直运行，直至它不得不放弃 CPU。

线程的调度不是跨平台的，它不仅仅取决于 java 虚拟机，还依赖于操作系统。在某些操作系统中，只要运行中的线程没有遇到阻塞，就不会放弃 CPU；在某些操作系统中，即使线程没有遇到阻塞，也会运行一段时间后放弃 CPU，给其它线程运行的机会。

java 的线程调度是不分时的，同时启动多个线程后，不能保证各个线程轮流获得均等的.