Java 并发编程面试

**一、基础知识**

**□并发编程的优缺点**

**1.为什么要使用并发编程（并发编程的优点）**

1、"摩尔定律" 失效，硬件的单元计算能力提升受限；硬件上提高了 CPU 的核数和个数。并发编程可以提升 CPU 的计算能力的利用率。

2、提升程序的性能，如：响应时间、吞吐量、计算机资源使用率等。

3、并发程序可以更好地处理复杂业务，对复杂业务进行多任务拆分，简化任务调度，同步执行任务。

**2.并发编程有什么缺点**

**优点:**

图像处理，遍历耗时，充分利用多核CPU资源，提高效率；复杂业务分块，更适合复杂业务。

**缺点:**

a.上下文切换，要存储当前状态，加载状态，耗时。  
尽量不使用锁，导致竞争锁的上下文切换(CAS);  
最少使用线程，避免线程浪费等待。

b.死锁，锁没法释放，导致阻塞的线程一直阻塞。  
避免一个线程获得多个锁。  
线程间加锁顺序尽量一致。  
使用定时锁

**3.并行和并发有什么区别？**

**并发**：是指多个线程任务在同一个CPU上快速地轮换执行，由于切换的速度非常快，给人的感觉就是这些线程任务是在同时进行的，但其实并发只是一种逻辑上的同时进行；

**并行**：是指多个线程任务在不同CPU上同时进行，是真正意义上的同时执行。

**4.什么是多线程，多线程的优劣？**

定义：多线程是指程序中包含多个流，即在一个程序中可以同时进行多个不同的线程来执行不同的任务

优点:

可以提高CPU的利用率，在多线程中，一个线程必须等待的时候，CPU可以运行其它线程而不是等待，这样就大大提高了程序的效率，也就是说单个程序可以创建多个不同的线程来完成各自的任务。

缺点:

线程也是程序，线程也需要占内存，线程也多内存也占的也多。

多线程需要协调和管理，所以需要CPU跟踪线程。

线程之间共享资源的访问会相互影响，必须解决禁用共享资源的问题。

**□**线程和进程区别

1.什么是线程和进程?

**线程**：进程中的一个控制单元，负责当前进程中的程序执行，一个进程至少有一个线程，一个进程可以运行多个线程，多个线程可以共享数据。

**进程**：内存中运行的运用程序，每个进程都有自己独立的内存空间，一个进程可以由多个线程

2.进程与线程的区别 ？

**根本区别：**进程是操作系统资源

分配的基本单元，而线程是处理器任务调度的和执行的基本单位。

**资源开销：**每个进程都有自己独立的代码和空间(程序上下文)，程序之间的切换会有较大的开销；线程可以看作轻量级的进程，同一类线程共享代码和数据空间，每个线程都有自己独立的运行栈和程序计数器(PC)，线程之间切换的开销小。

**包含关系：**如果一个进程内有多个线程，则执行的过程不是一条线的，而是多条线（多个线程），共同完成；线程是进程的一部分，可以把线程看作是轻量级的进程。

**内存分配：**同一进程的线程共享本进程的地址空间和资源，而进程之间的地址空间和资源是相互独立的。

3.什么是上下文切换?

当前任务执行完，CPU时间片切换到另一个任务之前会保存自己的状态，以便下次再切换会这个任务时可以继续执行下去，任务从保存到再加载执行就是一次上下文切换。

4.守护线程和用户线程有什么区别呢？

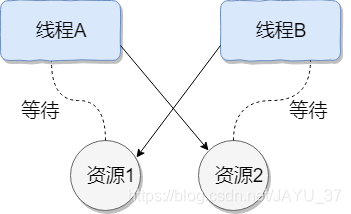
守护线程(Daemon)和用户线程(User)。

唯一的区别是判断虚拟机(JVM)何时离开，Daemon是为其他线程提供服务，如果全部的User Thread已经结束，Daemon 没有可服务的线程，JVM关闭。

5. 什么是线程死锁

死锁是指两个或两个以上进程(线程)在执行过程中，由于竞争资源或由于彼此通信造成的一种堵塞的现象，若无外力的作用下，都将无法推进，此时的系统处于死锁状态。

如图，线程A拥有的资源2，线程B拥有的资源1，此时线程A和线程B都试图去拥有资源1和资源2，但是它们的🔒还在，因此就出现了死锁。



6.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？

new Thread，线程进入了新建的状态，start方法的作用是使线程进入就绪的状态，当分配到时间片后就可以运行了。start方法会执行线程前的相应准备工作，然后在执行run方法运行线程体，这才是真正的多线程工作。

如果直接执行了run方法，run方法会被当作一个main线程下的普通方法执行，并不会在某个线程中去执行它，所以这并不是多线程工作。

7.什么是 Callable 和 Future?

Callable接口也类似于Runnable接口，但是Runnable不会接收返回值，并且无法抛出返回结果的异常，而Callable功能更强大，被线程执行后，可有返回值，这个返回值可以被Future拿到，也就是说Future可以拿到异步执行任务的返回值。

Future接口表示异步任务，是一个可能没有完成的异步任务结果，所以说Callable用于产生结果，Future用于接收结果。

□线程的状态和基本操作

1. 说说线程的生命周期及五种基本状态？

*很多地方说线程有5种状态，但实际上是6中状态，可以参考Thread类的官方api*

**新创建：**又称初始化状态，这个时候Thread才刚刚被new出来，还没有被启动。

**可运行状态：**表示已经调用Thread的start方法启动了，随时等待CPU的调度，此状态又被称为就绪状态。

**被终止：**死亡状态，表示已经正常执行完线程体run()中的方法了或者因为没有捕获的异常而终止run()方法了。

**计时状态：**调用sleep(参数)或wait(参数)后线程进入计时状态，睡眠时间到了或wait时间到了，再或者其它线程调用notify并获取到锁之后开始进入可运行状态。另一种情况，其它线程调用notify没有获取到锁或者wait时间到没有获取到锁时，进入堵塞状态。

**无线等待状态**：获取锁对象后，调用wait()方法，释放锁进入无线等待状态

**锁堵塞状态：**wait(参数)时间到或者其它线程调用notify后没有获取到锁对象都会进入堵塞状态，只要一获取到锁对象就会进入可运行状态。

2.Java 中用到的线程调度算法是什么？

有两种调度模型：分时调度和抢占式调度

分时调度就是让所有的线程轮流获得CPU的使用权，并且平均分配到各个线程占有CPU的时间片。

抢占式调度：Java虚拟机采用抢占式调度模型，是指优先让线程池中优先级高的线程首先占用CPU，如果线程池中优先级相同，那么随机选择一个线程，使其占有CPU，处于这个状态的CPU会一直运行，优先级高的分的CPU的时间片相对会多一点。

3.线程的调度策略

线程调度优先选择优先级高的运行，但是如果出现一下情况，就会终止运行(不是进入死亡状态)：

1. 线程调用了yield方法让出CPU的使用权，线程进入就绪状态。
2. 线程调用sleep()方法，使其进入计时状态
3. 线程由于IO受阻
4. 另一个更高的优先级线程出现
5. 在支持的时间片系统中，改线程的时间片用完。

4. 请说出与线程同步以及线程调度相关的方法。

wait()：调用后线程进入无限等待状态，并释放所持对象的锁

sleep()：使一个线程进入休眠状态（堵塞状态），带有对象锁，是一个静态方法，需要处理InterruptException异常。

notify()：唤醒一个处于等待状态的线程(无线等待或计时等待)，如果多个线程在等待，并不能确切的唤醒一个线程，与JVM确定唤醒那个线程，与其优先级有关。

notityAll()：唤醒所有处于等待状态的线程，但是并不是将对象的锁给所有的线程，而是让它们去竞争，谁先获取到锁，谁先进入就绪状态。

6. sleep() 和 wait() 有什么区别？

两者都可以使线程进入等待状态

类不同：sleep()是Thread下的静态方法，wait()是Object类下的方法

* 是否释放锁：sleep()不释放锁，wait()释放锁
* 用处不同：wait()常用于线程间的通信，sleep()常用于暂停执行。
* 用法不同：wait()用完后，线程不会自动执行，必须调用notify()或notifyAll()方法才能执行，sleep()方法调用后，线程经过过一定时间会自动苏醒，wait(参数)也可以传参数使其苏醒。它们苏醒后还有所区别，因为wait(）会释放锁，所以苏醒后没有获取到锁就进入堵塞状态，获取到锁就进入就绪状态，而sleep苏醒后之间进入就绪状态，但是如果cpu不空闲，则进入的是就绪状态的堵塞队列中。

7.为什么 wait(), notify()和 notifyAll()必须在同步方法或者同步 块中被调用？

Java中任何对象都可以被当作锁对象，wait(),notify(),notifyAll()方法用于等待获取唤醒对象去获取锁，Java中没有提供任何对象使用的锁，但是任何对象都继承于Object类，所以定义在Object类中最合适。

8. Thread 类中的 yield 方法有什么作用？

让出CPU的使用权，使当前线程从运行状态进入就绪状态，等待CPU的下次调度。

9.如何停止一个正在运行的线程？

1. 使用stop方法终止，但是这个方法已经过期，不被推荐使用。
2. 使用interrupt方法终止线程
3. run方法执行结束，正常退出

10.什么是阻塞式方法？

**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。**

11.Java 中你怎样唤醒一个阻塞的线程？

**1. sleep() 方法**

sleep（毫秒），指定以毫秒为单位的时间，使线程在该时间内进入线程阻塞状态，期间得不到cpu的时间片，等到时间过去了，线程重新进入可执行状态。（暂停线程，不会释放锁）

**2.suspend() 和 resume() 方法**

挂起和唤醒线程，suspend e()使线程进入阻塞状态，只有对应的resume e()被调用的时候，线程才会进入可执行状态。（不建议用，容易发生死锁）

**3. yield() 方法**

会使的线程放弃当前分得的cpu时间片，但此时线程任然处于可执行状态，随时可以再次分得cpu时间片。yield()方法只能使同优先级的线程有执行的机会。调用 yield()的效果等价于调度程序认为该线程已执行了足够的时间从而转到另一个线程。（暂停当前正在执行的线程，并执行其他线程，且让出的时间不可知）

**4.wait() 和 notify() 方法**

两个方法搭配使用，wait()使线程进入阻塞状态，调用notify()时，线程进入可执行状态。wait()内可加或不加参数，加参数时是以毫秒为单位，当到了指定时间或调用notify()方法时，进入可执行状态。（属于Object类，而不属于Thread类，wait()会先释放锁住的对象，然后再执行等待的动作。由于wait()所等待的对象必须先锁住，因此，它只能用在同步化程序段或者同步化方法内，否则，会抛出异常IllegalMonitorStateException.）

**5.join()方法**

也叫线程加入。是当前线程A调用另一个线程B的join()方法，当前线程转A入阻塞状态，直到线程B运行结束，线程A才由阻塞状态转为可执行状态。

12. notify() 和 notifyAll() 有什么区别？

notify() 方法随机唤醒对象的等待池中的一个线程，进入锁池；notifyAll() 唤醒对象的等待池中的所有线程，进入锁池。

*等待池：假设一个线程A调用了某个对象的wait()方法，线程A就会释放该对象的锁后，进入到了该对象的等待池，等待池中的线程不会去竞争该对象的锁。*

*锁池：只有获取了对象的锁，线程才能执行对象的 synchronized 代码，对象的锁每次只有一个线程可以获得，其他线程只能在锁池中等待*

13. 如何在两个线程间共享数据？

两个线程之间共享变量即可实现共享数据。  
一般来说，共享变量要求变量本身是线程安全的，然后在线程中对变量使用。

14.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，因为它不会锁着整个对象，当然你也可以然它锁住整个对象。同步方法会锁住整个对象，哪怕这个类中有不关联的同步块，这通常会导致停止继续执行，并等待获取这个对象锁。  
同步块扩展性比较好，只需要锁住代码块里面相应的对象即可，可以避免死锁的产生。  
**原则**：同步范围也小越好。

15. 什么叫线程安全？servlet 是线程安全吗?

线程安全是指某个方法在多线程的环境下被调用时，能够正确处理多线程之间的共享变量，能程序能够正确完成。  
Servlet不是线程安全的，它是单实例多线程的，当多个线程同时访问一个方法时，不能保证共享变量是安全的。

16. 在 Java 程序中怎么保证多线程的运行安全？

1. 使用安全类，比如 java.util.concurrent 下的类，使用原子类AtomicInteger
2. 使用自动锁，synchronized锁
3. Lock lock = new ReentrantLock()，使用手动锁lock .lock()，lock.unlock()方法

17. 你对线程优先级的理解是什么？

每个线程都具有优先级的，一般来说，高优先级的在线程调度时会具有优先被调用权。我们可以自定义线程的优先级，但这并不能保证高优先级又在低优先级前被调用，只是说概率有点大。

线程优先级是1-10，1代表最低，10代表最高。

Java的线程优先级调度会委托操作系统来完成，所以与具体的操作系统优先级也有关，所以如非特别需要，一般不去修改优先级。

18. Java 线程数过多会造成什么异常？

1. 线程的生命周期开销非常高
2. 消耗过多的CPU 资源
3. 如果可运行的线程数量多于可用处理器的数量，那么有线程将会被闲置。大量空闲的线程会占用许多内存，给垃圾回收器带来压力，而且大量的线程在竞争CPU 资源时还将产生其他性能的开销。
4. 降低稳定性

JVM 在可创建线程的数量上存在一个限制，这个限制值将随着平台的不同而不同，并且承受着多个因素制约，包括JVM 的启动参数、Thread 构造函数中请求栈的大小，以及底层操作系统对线程的限制等。如果破坏了这些限制，那么可能抛出OutOfMemoryError 异常

二、并发理论

□Java 内存模型

1. Java 中垃圾回收有什么目的？什么时候进行垃圾回收？

垃圾回收是在内存中存在没有引用的对象或超过作用域的对象时进行的。

垃圾回收的目的是识别并且丢弃应用不再使用的对象来释放和重用资源。

1. 如果对象的引用被置为 null，垃圾收集器是否会立即释放对象占 用的内存？

不会，在下一个垃圾回调周期中，这个对象将是被可回收的。

也就是说并不会立即被垃圾收集器立刻回收，而是在下一次垃圾回收时才会释放其占用的内存。

1. finalize()方法什么时候被调用？析构函数(finalization)的目的 是什么？

垃圾回收器（garbage colector）决定回收某对象时，就会运行该对象的finalize()方法；

finalize是Object类的一个方法，该方法在Object类中的声明protected void finalize() throws Throwable { }

在垃圾回收器执行时会调用被回收对象的finalize()方法，可以覆盖此方法来实现对其资源的回收。

*注意：一旦垃圾回收器准备释放对象占用的内存，将首先调用该对象的finalize()方法，并且下一次垃圾回收动作发生时，才真正回收对象占用的内存空间*

GC本来就是内存回收了，大部分时候，什么都不用做(也就是不需要重载)。只有在某些很特殊的情况下，比如你调用了一些native的方法(一般是C写的)，可以要在finaliztion里去调用C的释放函数。

1. synchronized 和 volatile 的区别是什么？

synchronized 是悲观锁，属于抢占式，会引起其他线程阻塞。

volatile 提供多线程共享变量可见性和禁止指令重排序优化。

1. 乐观锁和悲观锁的理解及如何实现，有哪些实现方式？

**悲观锁**

总是假设最坏的情况，每次取数据时都认为其他线程会修改，所以都会加锁（读锁、写锁、行锁等），当其他线程想要访问数据时，都需要阻塞挂起。可以依靠数据库实现，如行锁、读锁和写锁等

**乐观锁**  
总是认为不会产生并发问题，每次去取数据的时候总认为不会有其他线程对数据进行修改，因此不会上锁，但是在更新时会判断其他线程在这之前有没有对数据进行修改，一般会使用版本号机制或CAS操作实现。

**悲观锁的实现**：  
1.传统的关系型数据库使用这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。  
2.Java 里面的 synchronized和ReentrantLock独占锁也属于悲观锁。

**乐观锁的实现**：  
CAS 实现：Java 中java.util.concurrent.atomic包下面的原子变量使用了乐观锁的一种 CAS 实现方式。

三、线程池

□ Executors 类创建四种常见线程池

1. 什么是线程池？有哪几种创建方式？

线程池就是创建若干个可执行的线程放入一个池（容器）中，有任务需要处理时，会提交到线程池中的任务队列，处理完之后线程并不会被销毁，  
而是仍然在线程池中等待下一个任务。

Executors工厂方法创建

自定义创建

{通过new ThreadPoolExecutor(int corePoolSize, int maximumPoolSize, long keepAliveTime, TimeUnit unit, BlockingQueue workQueue)}

1. 你知道怎么创建线程池吗？

**Java通过Executors（jdk1.5并发包）提供四种线程池，分别为：**

**newCachedThreadPool创建一个可缓存线程池，如果线程池长度超过处理需要，可灵活回收空闲线程，若无可回收，则新建线程。**

**newFixedThreadPool 创建一个定长线程池，可控制线程最大并发数，超出的线程会在队列中等待。**

**newScheduledThreadPool 创建一个定长线程池，支持定时及周期性任务执行。**

**newSingleThreadExecutor 创建一个单线程化的线程池，它只会用唯一的工作线程来执行任务，保证所有任务按照指定顺序(FIFO, LIFO, 优先级)执行。**

1. ThreadPoolExecutor 构造函数重要参数分析

1. **corePoolSize**。核心线程池大小。这个参数是否生效取决于allowCoreThreadTimeOut变量的值，该变量默认是false，即对于核心线程没有超时限制，所以这种情况下，corePoolSize参数是起效的。

2. **maximumPoolSize**。线程池中最大的存活线程数。这个参数比较好理解，对于超出corePoolSize部分的线程，无论allowCoreThreadTimeOut变量的值是true还是false，都会超时，超时时间由keepAliveTime和unit两个参数算出。

3. **keepAliveTime**。超时时间。

4. **unit**。超时时间的单位，秒，毫秒，微秒，纳秒等，与keepAliveTime参数共同决定超时时间。

5. **workQueue**。当调用execute方法时，如果线程池中没有空闲的可用线程，那么就会把这个Runnable对象放到该队列中。这个参数必须是一个实现BlockingQueue接口的阻塞队列，因为要保证线程安全。

6. **threadFactory**。线程工厂类。用于在需要的时候生成新的线程。默认实现是Executors.defaultThreadFactory()，即new 一个Thread对象，并设置线程名称，daemon等属性。

7. **handler**。这个参数的作用是当提交任务时既没有空闲线程，任务队列也满了，这时候就会调用handler的rejectedExecution方法。默认的实现是抛出一个RejectedExecutionException异常。

1. ThreadPoolExecutor 饱和策略

当有界队列被填满后，饱和策略开始发挥作用。ThreadPoolExecutor 的饱和策略可以通过调用 setRejectedExecutionHandler 来修改。

JDK 提供了**四**种不同的饱和策略，实现 RejectedExecutionHandler 接口。

### AbortPolicy 中止策略

**中止策略**，默认的饱和策略，该策略将抛出未检查的 RejectedExecutionException。调用者可以捕获这个异常，然后根据需求编写自己的处理代码。

### DiscardPolicy 抛弃策略

**抛弃策略**，会悄悄抛弃该任务，当新提交的任务无法保存到队列中等待执行时。

**DiscardOldestPolicy 抛弃最旧的策略**

抛弃最旧的策略，会抛弃下一个将被执行的任务，然后尝试重新提交新的任务（如果工作队列是一个优先队列，那么该策略将导致抛弃优先级最高的任务，因此最好不要将该策略和优先级队列 (PriorityBlockingQueue)放在一起使用）。

### CallerRunsPolicy 调用者运行策略

**调用者运行策略**，实现了一种调节机制，该策略既不会抛弃任务，也不会抛出异常，而是将某些任务回退到调用者，从而降低新任务的流量。它不会在线程池中执行新的提交任务，而是在一个调用了 execute 的线程中执行该任务。

⑤一个简单的线程池 Demo:`Runnable`+`ThreadPoolExecutor`

首先创建一个Runnable类

MyRunnable.java

编写测试程序， ThreadPoolExecutor构造自定义参数的方式来创建线程池

ThreadPoolExecutor.java