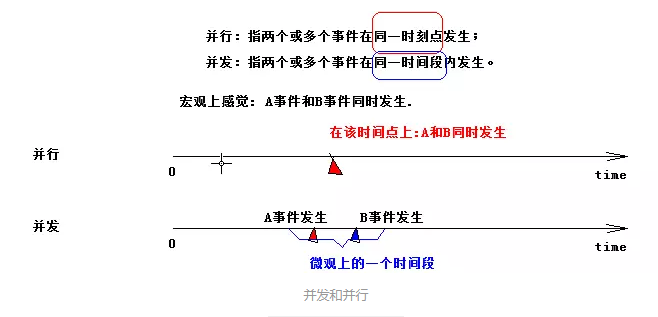
## 并发基础

**1.并行和并发**

**并行性是指两个或多个事件在同一时刻发生。而并发性是指两个或多个事件在同一时间间隔内发生。**例如，在1秒时间内，0-15ms程序A运行，15-30ms程序B运行，30-45ms程序C运行，45-60ms程序D运行，因此可以说，在1s时间间隔内，宏观上有四道程序在同时运行，但微观上，程序A、B、C、D是分别交替进行的。



如果在计算机系统中有多个处理机，这些可以并发执行的程序就可以分配到多个处理机上，实现并发执行，即利用每个处理机处理一个可并发的程序。这样，多个程序便可以同时执行。依次提高系统的资源利用率，增加系统的吞吐量。

**2.进程和线程**

**进程是指一个内存中运行的应用程序**。一个应用程序可以同时启动多个进程，我们启动两个进程，一个用来打游戏，一个用来播放音乐。但是进程本身就是非常消耗资源的东西，同时进程之间的通信很不方便，引入“线程”。

**线程是指进程中的一个执行任务（控制单元），一个进程可以同时并发运行多个线程。**

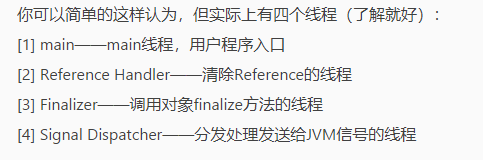
**3.进程和线程的区别**

**进程**：有独立的内存空间，进程中的数据存放空间（堆空间和栈空间）是独立的，至少有一个线程；

**线程**：堆空间是共享的，栈空间是独立的，线程消耗的资源也比进程小，相互之间可以影响的，又称为轻型进程或者进程元。

因为一个进程中的多个线程是并发运行的，那么从微观角度上考虑也是有先后顺序的，那么**哪个线程执行完全取决于CPU调度（JVM来调度）**，程序员是控制不了的。我们可以把多线程并发性看作是多个线程在瞬间抢夺CPU资源，谁抢到资源谁就运行，这也就造成了**多线程的随机性**。

**Java程序的进程（Java的一个程序运行在系统中）里至少包含主线程和垃圾回收线程（后台线程）：**



**4.多线程的优势**：

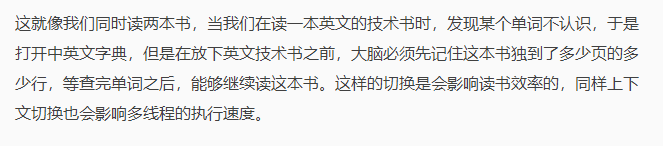
①资源利用率更好；（CPU能够在等待IO的时候做一些其他的事情）

②程序设计在某些情况下更简单；

③程序响应更快。

**5.上下文切换**

CPU通过给每个线程分配CPU时间片，时间片是CPU分配给各个线程的时间，因为时间片非常短，所以CPU通过不停的切换线程执行，让我门感觉多个线程是同时执行的，时间片一般是几十毫秒。CPU通过时间片分配算法来循环执行任务，当前任务执行一个时间片后切换到下一个任务。**但是**，在切换前会保存上一个任务的状态，以便下次切换回这个任务的时候，可以再加载这个任务的状态。**所以任务从保存到再加载的过程就是一次上下文切换。**

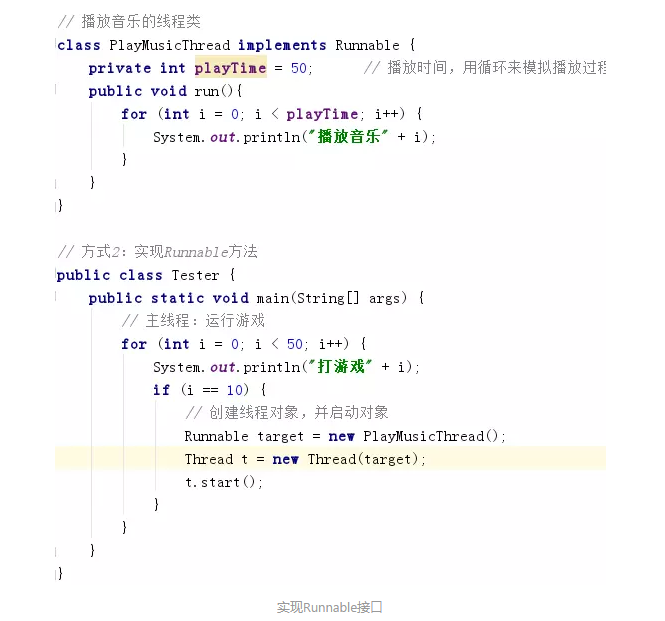


**6.创建线程的两种方式：**

①继承Thread类（extends）：

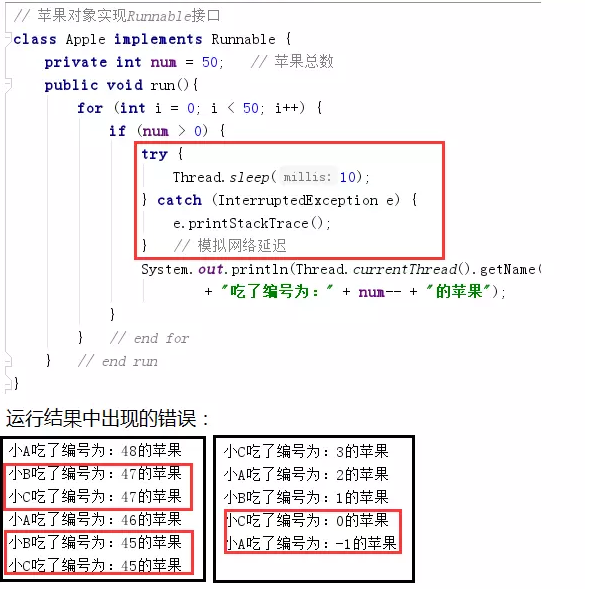


②实现Runnable接口（implements）：



**7.线程不安全问题**

因为没有任何操作来限制线程**获取相同资源**并对它们进行操作。

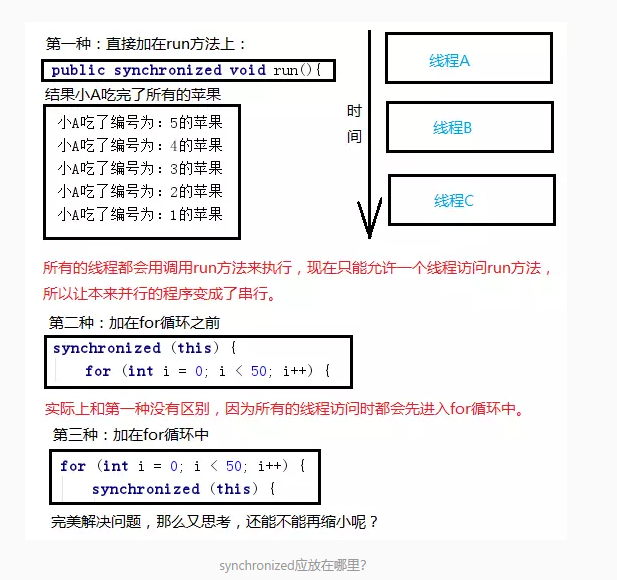


**8.线程同步**

多个执行线程共享一个资源的情景，为了解决访问共享资源错误或数据不一致的问题，引入临界区的概念：**用来访问共享资源的代码块，这个代码块在同一时间内只允许一个线程执行**。

**9.synchronized关键字**

如果一个对象用synchronized关键字声明，那**么只有一个执行线程允许访问它**。使用synchronized的好处：保证了多线程并发访问时的同步操作，避免线程的安全性问题。坏处：使用synchronized的方法/代码块的性能比不用要低一些。所以好的做法是：尽量减小synchronized的作用域。



**10.同步锁（Lock）**

Lock机制提供了比synchronized代码块和synchronized方法更广泛的锁定操作，同步代码块/同步方法具有的功能Lock都有，除此之外更强大，更体现面向对象。



## 高并发编程基础知识

**1.多线程和单线程的区别？**

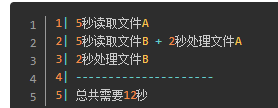
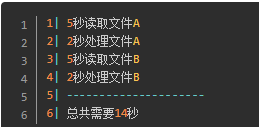
答：①在单核CPU中，将CPU分为很小的**时间片**，在每一时刻只能有一个线程在执行，是一种微观上轮流占用CPU的机制。②**多线程**会存在线程上下文切换，会导致程序执行速度变慢，即采用一个拥有两个线程的进程执行所需要的时间比一个线程的进程执行两次所需要的时间要多一些。

**结论：**即采用多线程不会提高程序的执行速度，反而会降低速度，但是对于用户来说，可以减少用户的响应时间。

**面试官：多线程的优势？**

答：**①资源利用率更好**

例如：一个应用程序需要从本地文件系统中**读取和处理文件**。比方说，从磁盘读取一个文件需要5秒，处理一个文件需要2秒，则处理两个文件需要的时间：



从磁盘读取文件的时候，大部分的CPU时间用来等待，处于空闲。通过改变操作顺序，就能够更好的使用CPU资源，CPU等待第一个文件被读取完，然后开始读取第二个文件，当第二个文件在被读取的时候，CPU会去处理第一个文件。记住，在**等待磁盘读取文件的时候，CPU大部分时间是空闲的**。

**②程序设计在某些情况下更简单**

**③程序响应更快**

**2.多线程一定快吗？**

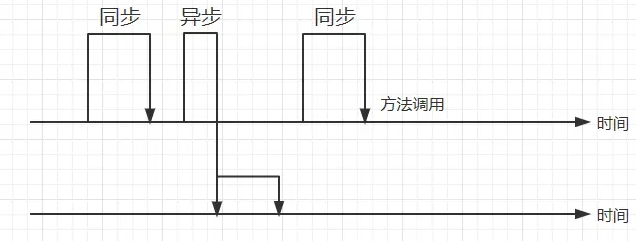
答：不一定。因为线程有创建和上下文切换的开销。

**3.什么是同步？什么是异步？**

答：同步和异步通常用来形容一次方法调用。

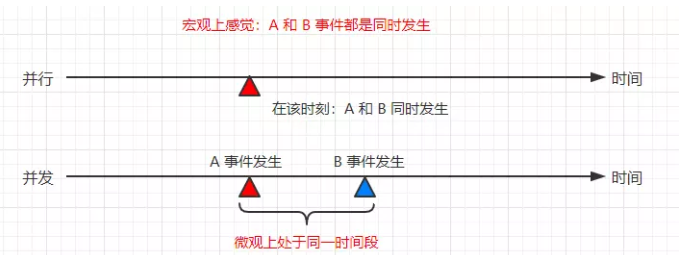
**同步方法调用一旦开始，调用者必须等方法返回后，才能继续后续的行为**。这就好像是我们去商城买一台空调，你看中了一台空调，于是就跟售货员下了单，然后售货员就去仓库帮你调配物品，这天你热的实在不行，就催着商家赶紧发货，于是你就在商店里等着，直到商家把你和空调都送回家，一次愉快的购物才结束，这就是同步调用。

**异步方法更像是一个消息传递，一旦开始，方法调用就会立即返回，调用者就可以继续后序的操作**。我们可以坐在里打开电脑，在网上订购一台空调。当你完成网上支付的时候，对你来说购物过程已经结束了。虽然空调还没有送到家，但是你的任务都已经完成了。商家接到你的订单后，就会加紧安排送货，当然这一切已经跟你无关了，你已经支付完成，想什么就能去干什么了，出去溜达几圈都不成问题。等送货上门的时候，接到商家电话，回家一趟签收即可。这就是异步调用



面试官：那并发（Concurrency）和并行（Parallelism）的区别呢？

**答：并行性是指两个或多个事件在同一时刻发生。而并发性是指两个或多个事件在同一事件间隔内发生。**



**4.线程和进程的区别（必考）**

答：①进程是一个“执行中的程序”，是系统进行资源分配和调度的一个独立单位；

②线程是进程的一个实体，一个进程中拥有多个线程，线程之间共享地址空间和其它资源（所以通信和同步等操作比进程更加容易）；

③线程上下文切换比进程上下文切换要快得多。进程切换时，涉及到当前进程的CPU环境的保存和新被调度运行进程的CPU环境的设置；线程切换仅需要保存和设置少量的寄存器内容，不涉及存储管理方面的操作。

**面试官：进程间如何通讯？线程间如何通讯？**

答：进程间通讯依靠IPC资源，例如管道（pipes）、套接字（sockets）等；

线程间通讯依靠JVM提供的API，例如wait（）、notify（）、notifyAll（）等方法，线程间还可以通过共享的主内存来进行值传递。

**5.什么时阻塞（Blocking）和非阻塞（Non-Blocking）？**

答：阻塞和非阻塞通常用来形容**多线程间**的相互影响。比如一个线程占用了临界区资源，那么其它所有需要这个资源的线程就必须在这个临界区中及进行等待。等待会导致**线程挂起**，这种情况就是阻塞。此时，如果占用资源的线程已知不愿意释放资源，那么其它所有阻塞在这个临界区上的线程都不能工作。

非阻塞的意思与之相反，它强调没有一个线程可以妨碍其它线程执行。所有的线程都会尝试不断前向执行。

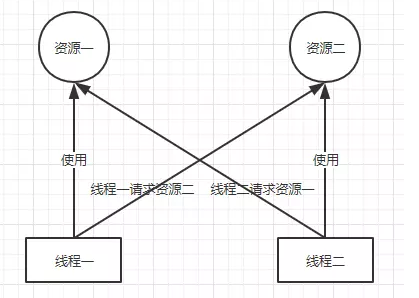
**面试官：临界区是什么？**

答：临界区用来表示**一种公共资源或者说是共享资源**，可以被多个线程使用。但是每一次，只能由一个线程使用它，一旦临界区资源被占用，其它线程要想使用这个资源，就必须等待。

比如比如，在一个办公室里有一台打印机，打印机一次只能执行一个任务。如果小王和小明同时需要打印文件，很显然，如果小王先下发了打印任务，打印机就开始打印小王的文件了，小明的任务就只能等待小王打印结束后才能打印，这里的打印机就是一个临界区的例子。在并行程序中，临界区资源是保护的对象，如果意外出现打印机同时执行两个打印任务，那么最可能的结果就是打印出来的文件就会是损坏的文件，它既不是小王想要的，也不是小明想要的。

**6.什么是死锁（Deadlock）、饥饿（Stavation）、活锁（Livelock）？**

答：死锁、饥饿和活锁都属于**多线程的活跃性问题**，如果发现上述几种情况，那么相关线程可能就不再活跃，也就说它可能很难再继续往下执行。



①死锁应该是最糟糕的一种情况了，它表**示两个或者两个以上的进程**在执行过程中，由于**竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去**。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

②饥饿是指某**一个或者多个线程因为种种原因无法获得所需要的资源，导致一直无法执行**。比如：1）它的线程**优先级**可能太低，而高优先级的线程不断抢占它需要的资源，导致低优先级的线程无法工作。2）某一个线程一直占着关键资源不放，导致其它需要这个资源的线程无法正常执行。

与死锁相比，**饥饿还是有可能在未来的一段时间内解决**，即它可能不需要外力作用，就可以正常执行下去（比如高优先级的线程已经完成任务，不再疯狂执行）

③活锁：线程之间都秉承着“谦让”的原则，**主动将资源释放给他人使用**，那么就会出现资源不断在两个线程中跳动，而没有一个线程可以同时拿到所有资源而征程执行。当你要坐电梯下楼，电梯到了，门开了，这时你正准备出去，但不巧的是，门外一个人挡着你的去路，他想进来。于是你很绅士的靠左走，避让对方，但同时对方也很绅士，但他靠右走希望避让你。结果，你们又撞上了。于是乎，你们都意识到了问题，希望尽快避让对方，你立即向右走，他也立即向左走，结果又撞上了！不过介于人类的只能，我相信这个动作重复 2、 3 次后，你应该可以顺利解决这个问题

**7.多线程产生死锁的四个必要条件？**

答： ①互斥条件：一个资源每次只能被一个线程使用；

②请求与保持条件：一个线程因请求资源而阻塞时，对已获得的资源保持不放；

③不剥夺条件：进程已经获得的资源，在未使用完之前，不能强行剥夺；

④循环等待条件：若干线程之间形成一种头尾相接的循环等待资源关系。

**面试官：如何避免死锁**？

答：**指定获取锁的顺序**，例如：

①比如某个线程只有获得A锁和B锁才能对某资源进行操作，在多线程条件下，如何避免死锁？

②获得锁的顺序是一定的，比如规定，只用获得A锁的线程才能有资格获取B锁，按顺序获取锁就可以避免死锁！！！

**8.如何指定多个线程的执行顺序？**

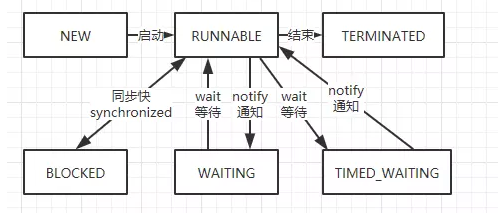
例如让10个线程按照顺序打印0123456789

答：①设定一个orderNum，每个线程执行结束后，更新orderNum，指明下一个要执行的线程。并且唤醒所有的等待线程。

②在每一线程的开始，要while判断orderNum是否等于自己的要求值（printNum），不是，则wait，是，则执行本线程。

**9.Java中线程有几种状态？**

答：六种，并且某个时刻Java线程只能处于其中的一种状态。



①新建（NEW）状态：表示**新创建**了一个线程对象，而此时**线程并没有开始执行**。

②可运行（RUNNABLE）状态：线程对象创建后，其它线程（比如main线程）调用了该对象的start（）/run（）方法，才表示线程开始执行。当线程执行时，处于RUNNBALE状态，**表示线程所需的一切资源已经准备好了**。该状态的线程位于可运行线程池中，等待被线程调度选中，获取cpu的使用权。

③阻塞（BLOCKED）状态：如果线程在执行过程终于**到了synchronized同步块**，就会进入BLOCKED阻塞状态，这时线程就会暂停执行，直到获得请求的锁。

④等待（WAITING）状态：当线程等待另一个线程通知调度器一个条件时，它自己进入等待状态。在**调用Object.wait方法或Thread.join方法**，或者是**等待java.util.concurrent库中的Lock或Condition**时，就会出现这种情况。

⑤计时等待（TIMED\_WATING）状态：Object.wait、Thread.join、Lock.tryLock和Condition.await 等方法**有超时参数**，还有 Thread.sleep 方法、LockSupport.parkNanos 方法和 LockSupport.parkUntil 方法，这些方法会导致线程进入计时等待状态，如果超时或者出现通知，都会切换会可运行状态。

⑥终止（TERMINATED）状态：当线程执行完毕，则进入该状态，表示结束。

**注意：**从 NEW 状态出发后，线程不能再回到 NEW 状态，同理，处于 TERMINATED 状态的线程也不能再回到 RUNNABLE 状态。

## 高并发编程—JUC包

JUC包—**java.util.concurrent包**，在此包中增加了并发编程中很常用的使用故居类，用于定义类似于线程的自定义子系统，包括线程池、异步IO和轻量级任务框架。

**1.sleep(n) 和wait（）的区别**？

答：①sleep方法：是**Thread类的静态方法**，当前线程将睡眠n毫秒，**线程进入阻塞状态**。当睡眠时间到了，会解除阻塞，**运行可执行状态**，等待CPU的到来。睡眠不释放锁（如果有的话）；

②wait方法：是**Object的方法，必须与synchronized关键字一起使用，线程进入阻塞状态**。当**notify或者notifyAll**被调用后，会解除阻塞。但是，只有重新占用互斥锁之后会进入可执行状态。**睡眠时，释放互斥锁**。

**2.synchronized关键字：**

答：**底层实现：**

①进入时，执行**monitorenter**，将**计数器**+1释放锁**monitorexit**时，计数器-1；

②当一个线程**判断到计数器为0时**，则**当前锁空闲**，可以占用；反之，当前线程进入等待状态。

**含义（monitor机制）：**synchronized是在**加锁，加对象锁**。对象锁是一种**重量锁**（monitor），synchronized的锁机制会根据**线程竞争情况在运行时**会有**偏向锁（单一进程）**、**轻量锁（多个线程访问sychronized区域）**、**对象锁（重量锁，多个线程存在竞争的情况）**、自旋锁等。

**3.volatile关键字**

答：该关键字可以**保证可见性不保证原子性**。

功能：①主存和工作内存：**直接与主存产生交互，进行读写操作，保证可见性（可见性，是指线程之间的可见性，一个线程修改的状态对另一个线程是可见的。也就是一个线程修改的结果。另一个线程马上就能看到。比如：用volatile修饰的变量，就会具有可见性。volatile修饰的变量不允许线程内部缓存和重排序，即直接修改内存。所以对其他线程是可见的。但是这里需要注意一个问题，volatile只能让被他修饰内容具有可见性，但不能保证它具有原子性。比如 volatile int a = 0；之后有一个操作 a++；这个变量a具有可见性，但是a++ 依然是一个非原子操作，也就是这个操作同样存在线程安全问题。）**；

②禁止JVM进行的指令重新排序。

**4.volatile能使得一个非原子操作变成原子操作吗？**

答：能。例如在类中有一个**long类型的成员变量**。如果你知道该成员变量会被多个线程访问，如计数器、价格等，你最好是将其设置为volatile。为什么？因为Java中读取long类型变量不是原子的，需要分为两步，如果一个线程正在修改该long变量的值，另一个线程可能**只能看到该值的一半**（前32位）。但是对一个volatile型的long或double变量的读写是原子。

**面试官：volatile修饰符有过什么实践吗？**

答：①一种实践是**用volatile修饰long和double变量，使其能按原子类型来读写**。double和long都是64位，因此对这两种类型的读是分两部分的，第一次读取第一个32位，然后再读剩下的32位，这个过程不是原子的，但Java中的volatile型的long或double变量的读写时原子的。

②volatile修饰符的另一个作用是**提供内存屏障**（memory barrier），例如在分布式框架中的应用。简单的说，就是当你写一个volatile变量之前，Java内存模型会插入一个写屏障（write barrier），读一个volatile之前，会插入一个读屏障（read barrier）。意思就是说，在你写一个 volatile 域时，能保**证任何线程都能看到你写的值**，同时，在写之前，也能**保证任何数值的更新对所有线程是可见的**，因为内存屏障会将其他所有写的值更新到缓存。

**5.ThreadLocal（线程局部变量）关键字**

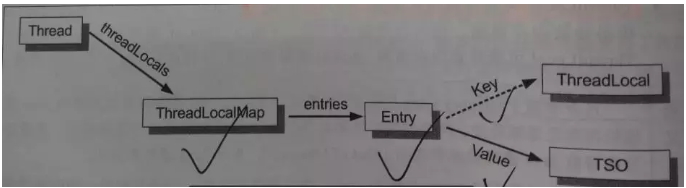
答：当使用ThreadLocal维护变量时，其**为每个使用该变量的线程提供独立的变量副本**，所以每一个线程都可以独立的改变自己的副本，而不会影响其他线程对应的副本。

ThreadLocal内部实现机制：

①每个线程内部都会维护一个类似HashMap的对象，称为**ThreadLocalMap**，里边会包含若干个Entry（Key-Value键值对），相应的线程被这些Entry的属主线程。

②Entry的**Key是一个ThreadLocal实例**，**Value是一个线程特有的对象**。Entry的作用即是：为其属主线程建立起一个ThreadLocal实例与一个线程特有对象之间的对应关系。

③Entry对Key的引用是**弱引用**；Entry对Value的引用是**强引用**。



**6.线程池有了解吗？（必考）**

答：**java.util.concurrent.ThreadPoolExecutor类**就是一个线程池（使得线程可以复用，就是执行完一个任务，并不被销毁，而是可以继续执行其他的任务）。**客户端调用ThreadPoolExecutor.submit(Runnable task)提交任务**，线程池内部维护的工作者**线程的数量**就是该线程池的线程池大小，有3中形态：

①当前线程池大小：表示线程池中实际工作者线程的数量；

②最大线程池大小（maximumPoolSize）：表示线程池中**允许存在**的工作者线程的数量上限；

③核心线程大小（corePoolSize）：表示一个不大于最大线程池大小的工作者线程数量上限。

**说明:**

①如果运行的线程**少于**corePoolSize，则Executor始终首选添加新的线程，而不进行排队；

②如果运行的线程**等于或者多于**corePoolSize，则Executor始终首选将请求加入队列，而不是添加新线程；

③如果**无法将请求加入队列**，即队列已经满了，则创建新的线程，除非创建此线程超出maximumPoolSize，在这种情况下，任务将被拒绝。

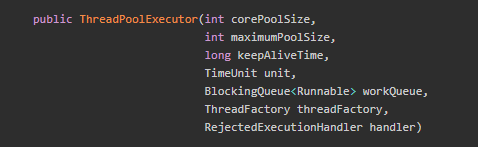
**面试官：我们为什么要使用线程池**？

答：①减少创建和销毁线程的次数，**每个工作线程都可以被重复利用**，可执行多个任务；

②可以根据系统的承受能力，**调整线程池中工作线程的数目，**放置因为消耗过多内存，而把服务器累趴下（每个线程大约需要1MB内存，现存开的越多，消耗的内存越大，最后死机）

**面试官：核心线程池内部实现了解吗？**

答：对于核心的几个线程池，无论是**newFixedThreadPool()**方法，**newSingleThreadExecutor（）**还是**newCachedThreadPool（）**方法，虽然看起来创建的线程有着完全不同的功能特点，但其实内部实现均使用来的ThreadPoolExecutor实现，其实都是**ThreadPoolExecutor类**的封装。



函数的参数含义如下：

①corePoolSize：指定了线程池中的线程数量；

②maximumPoolSize：指定了线程池中的最大线程数量。

③keepAliveTime：当线程池线程数量超过corePoolSize时，多余的空闲线程的存活时间。即，超过corePoolSize的空闲线程，在多长时间内，会被销毁。

④unit：keepAliveTime的单位。

⑤workQueue：任务队列，被提交但尚未被执行的任务。

⑥threadFactory：线程工厂，用于创建线程，一般用默认的即可；

⑦handler：拒绝策略。当任务太多来不及处理，如何拒绝任务。

**7.Atomic关键字**

答：可以使基本数据类型以原子的方式实现**自增自减**等操作。

**8.创建线程有哪几种方式？（必考）**

答：有两种创建线程的方式：一是**实现Runnable接口（implements）**，然后将它传递给Thread的构造函数，创建一个Thread对象；二是直接**继承Thread类（extends）**。

**面试官：两种方式有什么区别？**

1.继承方式：

①Java中类是**单继承**的，如果继承了Thread，该类就不能再有其他的父类了；

②从操作上分析，继承方式更简单，获取线程名字也简单；

③**从多线程共享同一个资源上分析，继承方式不能做到**。

2.实现方式：

①Java中类可以**实现多接口**，此时该类还可以继承其他类，并且还可以实现其他接口；

②从操作上分析，实现方式稍微复杂点，获取线程名字也比较复杂，得使用Thread.currentThread()来获取当前线程得引用；

③**从多线程共享同一个资源上分析，实现方式可以做到**（是否共享同一个资源）。

**9.run()方法和start（）方法有什么区别？**

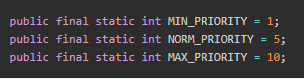
答：start（）方法（**继承方法创建线程的运行**）会**新建一个线程**并让这个线程执行run（）方法；而直接调用run（）方法只是作为一个普通的方法调用而已，它只会在当前进程中，串行执行run（）中的代码。

**10.你怎么理解线程优先级？**

答：Java中的线程可以有自己的优先级。**优先级高的线程在竞争资源时会更有优势**，更可能抢占资源，当然，这只是一个概率问题。如果运气不好，高优先级线程可能也会抢占失败。

由于线程的**优先级调度和底层操作系统**有密切的关系，在各个平台上表现不一，并且这种优先级产生的后果也可能不容易预测，无法精准控制，比如一个低优先级的线程可能一直抢占不到资源，从而始终无法运行，而产生饥饿（虽然优先级低，但是也不能饿死它啊）。因此，在要求严格的场合，还是**需要自己在应用层解决线程调度的问题**。

在Java中，使用1到10表示线程优先级，一般可以使用内置的三个静态标量表示：



**数字越大则优先级越高，但有效范围在 1 到 10 之间，默认的优先级为5**。

**11.在Java中如何停止一个线程？**

答：Java提供了丰富的API，但**没有为停止线程提供API**。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。

**12.多线程中得忙循环是什么？**

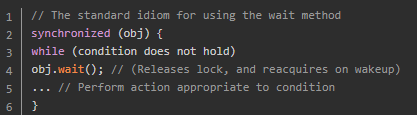
答：忙循环就是**程序员用循环让一个线程等待**，不像传统方法wait（）、sleep()或yield（）,它们都放弃了CPU控制权，而**忙循环不会放弃CPU，它就是在运行一个空循环。这么做的目的是为了保留CPU缓存**。在多核系统中，一个等待线程醒来的时候可能**会在另一个内核**运行，这样会重建缓存，为了**避免重建缓存和减少等待重建的时间**就可以使用它了。

**13.10个线程和2个线程的同步代码，哪个更容易写？**

答：从写代码的角度来说，两者的复杂度是相同的，因为同步代码与线程数量是相互独立的。但是**同步策略的选择**依赖于线程的数量，因为越多的线程意味着更大的竞争，所以你需要利用同步技术，如锁分离，这要求更复杂的代码和专业知识。

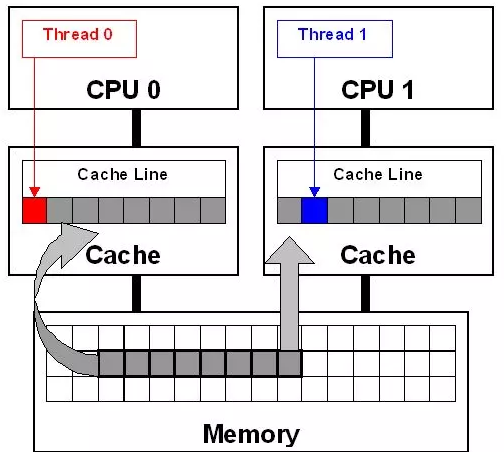
**14.你是如何调用wait（）方法的？使用if块还是循环？为什么？**

答：wait（）方法应该**在循环调用**，因为当线程获取到CPU开始执行的时候，其他条件可能还没有满足，所以在处理前，**循环检测条件是否满足**会更好。



**15.什么是多线程环境下的伪共享（false sharing）？**

答：伪共享是多线程系统（每个处理器有自己的局部缓存）中一个中所周知的性能问题。伪共享**发生在不同处理器上的线程对变量的修改依赖于相同的缓存行**。



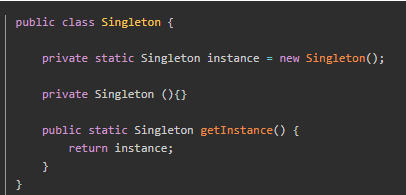
伪共享问题很难被发现，因为**线程可能访问完全不同的全局变量，内存中却碰巧在很相近的位置上**。如其他诸多的并发问题，避免伪共享的最基本方式是仔细审查代码，根据缓存行来调整你的数据结构。

**16.用wait-notify写一段代码来解决生产者-消费者问题（已解决）**

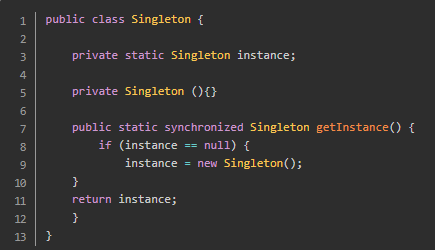
**17.用Java写一个线程安全的单例模式（Singleton）？**

解析：方法很多，但重点掌握的是**双重校验锁**。

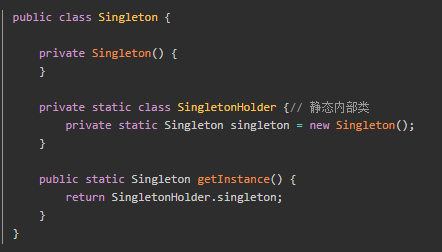
答：①饿汉式单例：饿汉式单例是指**在方法调用前，实例就已经创建好了**。下面是实现代码：



②加入synchronized的懒汉式单例：所谓懒汉式单例模式就是**在调用的时候才去创建这个实例**，我们在对外的创建实例**方法上加synchronized关键字**保证其在多线程中很好的工作。

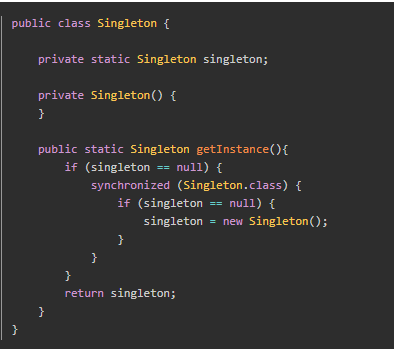


③使用静态内部类的方式创建单例：这种方**式利用了classloder的机制来保证初始化instance时只有一个线程**，它跟饿汉式的区别是：饿汉式只要Singleton类被加载了，那么instance就会被实例化（没有达到lazy loading的效果），而这种方式是**Singleton类被加载了，instance不一定被初始化**。只有显式通过**调用getInstance（）方法**时才会显示装载SingletonHoder类，从而实例化singleton。



**④双重校验锁**

为了达到线程安全，又能提高代码执行效率，我们这里可以采用**DCL的双检查锁机制**来完成，代码实现如下：



这种是用**双重判断来创建一个单例**的方法，那么我们为什么要使用两个if判断这个对象当前是不是空的呢 ？因为当有**多个线程同时要创建对象**的时候，多个线程有可能都停止在第一个if判断的地方**，等待锁的释放**，然后多个线程就都创建了对象，这样就不是单例模式了，所以我们要用两个if来进行这个对象是否存在的判断。