1. **JAVA**中的几种基本数据类型是什么，各自占用多少字节。
2. **String**类能被继承吗，为什么。
3. **String**，**Stringbuffer**，**StringBuilder**的区别。
4. **ArrayList**和**LinkedList**有什么区别。
5. 讲讲类的实例化顺序，比如父类静态数据，构造函数，字段，子类静态数据，构造函数，字

段，当**new**的时候，他们的执行顺序。

1. 用过哪些**Map**类，都有什么区别，**HashMap**是线程安全的吗,并发下使用的**Map**是什么，他们

内部原理分别是什么，比如存储方式，**hashcode**，扩容，默认容量等。

1. **JAVA8**的**ConcurrentHashMap**为什么放弃了分段锁，有什么问题吗，如果你来设计，你如何

设计。

1. 有没有有顺序的**Map**实现类，如果有，他们是怎么保证有序的。
2. 抽象类和接口的区别，类可以继承多个类么，接口可以继承多个接口么,类可以实现多个接口

么。

1. 继承和聚合的区别在哪。
2. **IO**模型有哪些，讲讲你理解的**nio** ，他和**bio**，**aio**的区别是啥，谈谈**reactor**模型。
3. 反射的原理，反射创建类实例的三种方式是什么。
4. 反射中，**Class.forName**和**ClassLoader**区别 。

5. 描述动态代理的几种实现方式，分别说出相应的优缺点。
6. 动态代理与**cglib**实现的区别。
7. 为什么**CGlib**方式可以对接口实现代理。
8. **final**的用途。
9. 写出三种单例模式实现 。
10. 如何在父类中为子类自动完成所有的**hashcode**和**equals**实现？这么做有何优劣。
11. 请结合**OO**设计理念，谈谈访问修饰符**public**、**private**、**protected**、**default**在应用设
12.

1、多线程有什么用？

1）发挥多核**CPU**的优势

2）防止阻塞

3）便于建模

2、创建线程的方式

1）继承Thread类

2）实现Runnable接口

3、start()方法和run()方法的区别

只有调用了start()方法，才会表现出多线程的特性，不同线程的run()方法里面的代码交替执行。如果只是调用run()方法，那么代码还是同步执行的，必须等待一个线程的run()方法里面的代码全部执行完毕之后，另外一个线程才可以执行其run()方法里面的代码。

4、Runnable接口和Callable接口的区别

Runnable接口中的run()方法的返回值是void，它做的事情只是纯粹地去执行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。

5、CyclicBarrier和CountDownLatch的区别

两个看上去有点像的类，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，只是给某个数值-1而已，该线程继续运行。

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。

6、volatile关键字的作用

1）多线程主要围绕可见性和原子性两个特性而展开，使用volatile关键字修饰的变量，保证了其在多线程之间的可见性，即每次读取到volatile变量，一定是最新的数据。

2）代码底层执行不像我们看到的高级语言----Java程序这么简单，它的执行是**Java代码-->字节码-->根据字节码执行对应的C/C++代码-->C/C++代码被编译成汇编语言-->和硬件**

电路交互，现实中，为了获取更好的性能JVM可能会对指令进行重排序，多线程下可能会出现一些意想不到的问题。使用volatile则会对禁止语义重排序，当然这也一定程度上降低了代码执行效率。

从实践角度而言，volatile的一个重要作用就是和CAS结合，保证了原子性，详细的可以参见java.util.concurrent.atomic包下的类，比如AtomicInteger，更多详情请点这里进行学习。

7、什么是线程安全

又是一个理论的问题，各式各样的答案有很多，我给出一个个人认为解释地最好的：如果你的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，那么你的代码就是线程安全的。

9、一个线程如果出现了运行时异常会怎么样

如果这个异常没有被捕获的话，这个线程就停止执行了。另外重要的一点是：如果这个线程持有某个某个对象的监视器，那么这个对象监视器会被立即释放

10、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

11、sleep方法和wait方法有什么区别

这个问题常问，sleep方法和wait方法都可以用来放弃CPU一定的时间，不同点在于如果线程持有某个对象的监视器，sleep方法不会放弃这个对象的监视器，wait方法会放弃这个对象的监视器

12、生产者消费者模型的作用是什么

这个问题很理论，但是很重要：

1)通过平衡生产者的生产能力和消费者的消费能力来提升整个系统的运行效率这是生产者消费者模型最重要的作用

2)解耦，这是生产者消费者模型附带的作用，解耦意味着生产者和消费者之间的联系少，联系越少越可以独自发展而不需要收到相互的制约

13、ThreadLocal有什么用

简单说ThreadLocal就是一种以空间换时间的做法，在每个Thread里面维护了一个以开地址法实现的ThreadLocal.ThreadLocalMap，把数据进行隔离，数据不共享，自然就没有线程安全方面的问题了

14、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()方法在调用前都必须先获得对象的锁

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

wait()方法和notify()/notifyAll()方法在放弃对象监视器的时候的区别在于 **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。

16、为什么要使用线程池

避免频繁地创建和销毁线程，达到线程对象的重用。另外，使用线程池还可以根据项目灵活地控制并发的数目。点击这里学习线程池详解。

17、怎么检测一个线程是否持有对象监视器

我也是在网上看到一道多线程面试题才知道有方法可以判断某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。

18、synchronized和ReentrantLock的区别

synchronized是和if、else、for、while一样的关键字，ReentrantLock是类，这是二者的本质区别。既然ReentrantLock是类，那么它就提供了比synchronized更多更灵活的特性，可以被继承、可以有方法、可以有各种各样的类变量，ReentrantLock比synchronized的扩展性体现在几点上：

(1)ReentrantLock可以对获取锁的等待时间进行设置，这样就避免了死锁

(2)ReentrantLock可以获取各种锁的信息

(3)ReentrantLock可以灵活地实现多路通知

另外，二者的锁机制其实也是不一样的。ReentrantLock底层调用的是Unsafe的park方法加锁，synchronized操作的应该是对象头中mark word，这点我不能确定。

19、ConcurrentHashMap的并发度是什么

ConcurrentHashMap的并发度就是segment的大小，默认为16，这意味着最多同时可以有16条线程操作ConcurrentHashMap，这也是ConcurrentHashMap对Hashtable的最大优势，任何情况下，Hashtable能同时有两条线程获取Hashtable中的数据吗？

20、ReadWriteLock是什么

首先明确一下，不是说ReentrantLock不好，只是ReentrantLock某些时候有局限。如果使用ReentrantLock，可能本身是为了防止线程A在写数据、线程B在读数据造成的数据不一致，但这样，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，没有必要加锁，但是还是加锁了，降低了程序的性能。

因为这个，才诞生了读写锁ReadWriteLock。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。

21、FutureTask是什么

这个其实前面有提到过，FutureTask表示一个异步运算的任务。FutureTask里面可以传入一个Callable的具体实现类，可以对这个异步运算的任务的结果进行等待获取、判断是否已经完成、取消任务等操作。当然，由于FutureTask也是Runnable接口的实现类，所以FutureTask也可以放入线程池中。

23、Java编程写一个会导致死锁的程序

第一次看到这个题目，觉得这是一个非常好的问题。很多人都知道死锁是怎么一回事儿：线程A和线程B相互等待对方持有的锁导致程序无限死循环下去。当然也仅限于此了，问一下怎么写一个死锁的程序就不知道了，这种情况说白了就是不懂什么是死锁，懂一个理论就完事儿了，实践中碰到死锁的问题基本上是看不出来的。

真正理解什么是死锁，这个问题其实不难，几个步骤：

1）两个线程里面分别持有两个Object对象：lock1和lock2。这两个lock作为同步代码块的锁；

2）线程1的run()方法中同步代码块先获取lock1的对象锁，Thread.sleep(xxx)，时间不需要太多，50毫秒差不多了，然后接着获取lock2的对象锁。这么做主要是为了防止线程1启动一下子就连续获得了lock1和lock2两个对象的对象锁

3）线程2的run)(方法中同步代码块先获取lock2的对象锁，接着获取lock1的对象锁，当然这时lock1的对象锁已经被线程1锁持有，线程2肯定是要等待线程1释放lock1的对象锁的

这样，线程1"睡觉"睡完，线程2已经获取了lock2的对象锁了，线程1此时尝试获取lock2的对象锁，便被阻塞，此时一个死锁就形成了。代码就不写了，占的篇幅有点多，Java多线程7：死锁这篇文章里面有，就是上面步骤的代码实现。

点击这里提供了一个死锁的案例。

24、怎么唤醒一个阻塞的线程

如果线程是因为调用了wait()、sleep()或者join()方法而导致的阻塞，可以中断线程，并且通过抛出InterruptedException来唤醒它；如果线程遇到了IO阻塞，无能为力，因为IO是操作系统实现的，Java代码并没有办法直接接触到操作系统。

25、不可变对象对多线程有什么帮助

前面有提到过的一个问题，不可变对象保证了对象的内存可见性，对不可变对象的读取不需要进行额外的同步手段，提升了代码执行效率。

27、如果你提交任务时，线程池队列已满，这时会发生什么

这里区分一下：

1）如果使用的是无界队列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

2）如果使用的是有界队列比如ArrayBlockingQueue，任务首先会被添加到ArrayBlockingQueue中，ArrayBlockingQueue满了，会根据maximumPoolSize的值增加线程数量，如果增加了线程数量还是处理不过来，ArrayBlockingQueue继续满，那么则会使用拒绝策略RejectedExecutionHandler处理满了的任务，默认是AbortPolicy

28、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。

29、Thread.sleep(0)的作用是什么

这个问题和上面那个问题是相关的，我就连在一起了。由于Java采用抢占式的线程调度算法，因此可能会出现某条线程常常获取到CPU控制权的情况，为了让某些优先级比较低的线程也能获取到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，这也是平衡CPU控制权的一种操作。

30、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，因为线程阻塞涉及到用户态和内核态切换的问题。既然synchronized里面的代码执行得非常快，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。

31、什么是Java内存模型

Java内存模型定义了一种多线程访问Java内存的规范。Java内存模型要完整讲不是这里几句话能说清楚的，我简单总结一下Java内存模型的几部分内容：

1)Java内存模型将内存分为了主内存和工作内存。类的状态，也就是类之间共享的变量，是存储在主内存中的，每次Java线程用到这些主内存中的变量的时候，会读一次主内存中的变量，并让这些内存在自己的工作内存中有一份拷贝，运行自己线程代码的时候，用到这些变量，操作的都是自己工作内存中的那一份。在线程代码执行完毕之后，会将最新的值更新到主内存中去

2)定义了几个原子操作，用于操作主内存和工作内存中的变量

3)定义了volatile变量的使用规则

4）happens-before，即先行发生原则，定义了操作A必然先行发生于操作B的一些规则，比如在同一个线程内控制流前面的代码一定先行发生于控制流后面的代码、一个释放锁unlock的动作一定先行发生于后面对于同一个锁进行锁定lock的动作等等，只要符合这些规则，则不需要额外做同步措施，如果某段代码不符合所有的happens-before规则，则这段代码一定是线程非安全的

33、什么是乐观锁和悲观锁

1）乐观锁：就像它的名字一样，对于并发间操作产生的线程安全问题持乐观状态，乐观锁认为竞争不总是会发生，因此它不需要持有锁，**比较-替换**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，那么就应该有相应的重试逻辑。

2）悲观锁：还是像它的名字一样，对于并发间操作产生的线程安全问题持悲观状态，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，不管三七二十一，直接上了锁就操作资源了。

点击这里了解更多乐观锁与悲观锁详情。

35、单例模式的线程安全性

老生常谈的问题了，首先要说的是单例模式的线程安全意味着某个类的实例在多线程环境下只会被创建一次出来。单例模式有很多种的写法，我总结一下：

1）饿汉式单例模式的写法：线程安全

2）懒汉式单例模式的写法：非线程安全

3）双检锁单例模式的写法：线程安全

36、Semaphore有什么作用

Semaphore就是一个信号量，它的作用是限制某段代码块的并发数。Semaphore有一个构造函数，可以传入一个int型整数n，表示某段代码最多只有n个线程可以访问，如果超出了n，那么请等待，等到某个线程执行完毕这段代码块，下一个线程再进入。由此可以看出如果Semaphore构造函数中传入的int型整数n=1，相当于变成了一个synchronized了。