多线程

1. java中有几种方法可以实现一个线程？     
           答：在Java中实现一个线程有两种方法，第一是实现Runnable接口实现它的run（）方法，第二种是继承Thread类，覆盖它的run（）方法。这两种方法的区别是，如果你的类已经继承了其它的类，那么你只能选择实现Runnable接口了，因为Java只允许单继承的。
2. 如何停止一个正在运行的线程？  
           答：当不阻塞时候设置一个标志位，让代码块正常运行结束并停止线程。如果发生了阻塞，用interupt()方法，Thread.interrupt（）方法不会中断一个正在运行的线程。这一方法实际上完成的是，在线程受到阻塞时抛出一个中断信号，这样线程就得以退出阻塞的状态。
3. notify()和notifyAll()有什么区别？  
           答：1）notify()和notifyAll()都是Object对象用于通知处在等待该对象的线程的方法。  
                  2）void notify(): 唤醒一个正在等待该对象的线程。  
                  3）void notifyAll(): 唤醒所有正在等待该对象的线程。  
           两者的最大区别在于：  
                  notifyAll使所有原来在该对象上等待被notify的线程统统退出wait的状态，变成等待该对象上的锁，一旦该对象被解锁，他们就会去竞争。  
            notify他只是选择一个wait状态线程进行通知，并使它获得该对象上的锁，但不惊动其他同样在等待被该对象notify的线程们，当第一个线程运            行完毕以后释放对象上的锁，此时如果该对象没有再次使用notify语句，即便该对象已经空闲，其他wait状态等待的线程由于没有得到该对象的          通知，继续处在wait状态，直到这个对象发出一个notify或notifyAll，它们等待的是被notify或notifyAll，而不是锁。
4. sleep()和 wait()有什么区别?  
           答：sleep是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用sleep不会释放对象锁。  
   wait是Object类的方法，对此对象调用wait方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出notify方法（或notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。
5. 什么是Daemon线程？它有什么意义？  
          答：所谓后台(daemon)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，并且这个线程并不属于程序中不可或缺的部分。因此，当所有的非后台线程介绍时，程序也就终止了，同时会杀死进程中的所有后台线程。反过来说，只要有任何非后台线程还在运行，程序就不会终止。必须在线程启动之前调用setDaemon()方法，才能把它设置为后台线程。注意：后台进程在不执行finally子句的情况下就会终止其run()方法。
6. java如何实现多线程之间的通讯和协作？  
         答： Java提供了3个非常重要的方法来巧妙地解决线程间的通信问题。这3个方法分别是：wait()、notify()和notifyAll()。它们都是Object类的最终方法，因此每一个类都默认拥有它们。虽然所有的类都默认拥有这3个方法，但是只有在synchronized关键字作用的范围内，并且是同一个同步问题中搭配使用这3个方法时才有实际的意义。这些方法在Object类中声明的语法格式如下所示：

**[html]** [view plain](http://blog.csdn.net/geolo/article/details/8670900) [copy](http://blog.csdn.net/geolo/article/details/8670900)

* 1. final void wait() throws InterruptedException
  2. final void notify()
  3. final void notifyAll()

锁

1. 什么是可重入锁（ReentrantLock）？

      答：  java.util.concurrent.lock 中的 Lock 框架是锁定的一个抽象，它允许把锁定的实现作为 Java 类，而不是作为语言的特性来实现。这就为 Lock 的多种实现留下了空间，各种实现可能有不同的调度算法、性能特性或者锁定语义。 ReentrantLock 类实现了 Lock ，它拥有与 synchronized 相同的并发性和内存语义，但是添加了类似锁投票、定时锁等候和可中断锁等候的一些特性。此外，它还提供了在激烈争用情况下更佳的性能。（换句话说，当许多线程都想访问共享资源时，JVM 可以花更少的时候来调度线程，把更多时间用在执行线程上。）  
          reentrant 锁意味着什么呢？简单来说，它有一个与锁相关的获取计数器，如果拥有锁的某个线程再次得到锁，那么获取计数器就加1，然后锁需要被释放两次才能获得真正释放。这模仿了 synchronized 的语义；如果线程进入由线程已经拥有的监控器保护的 synchronized 块，就允许线程继续进行，当线程退出第二个（或者后续）synchronized 块的时候，不释放锁，只有线程退出它进入的监控器保护的第一个 synchronized 块时，才释放锁。

1. 当一个线程进入某个对象的一个synchronized的实例方法后，其它线程是否可进入此对象的其它方法？  
            答：A、一个线程在访问一个对象的同步方法时，另一个线程可以同时访问这个对象的非同步方法  
                   B、 一个线程在访问一个对象的同步方法时，另一个线程不能同时访问这个同步方法。
2. synchronized和java.util.concurrent.locks.Lock的异同？  
            答：Lock 和 synchronized 有一点明显的区别 —— lock 必须在 finally 块中释放。否则，如果受保护的代码将抛出异常，锁就有可能永远得不到释放！这一点区别看起来可能没什么，但是实际上，它极为重要。忘记在 finally 块中释放锁，可能会在程序中留下一个定时炸弹，当有一天炸弹爆炸时，您要花费很大力气才有找到源头在哪。而使用同步，JVM 将确保锁会获得自动释放。
3. 乐观锁和悲观锁的理解及如何实现，有哪些实现方式？

并发框架

1. SynchronizedMap和ConcurrentHashMap有什么区别？  
           答：java5中新增了ConcurrentMap接口和它的一个实现类ConcurrentHashMap。ConcurrentHashMap提供了和Hashtable以及SynchronizedMap中所不同的锁机制。比起synchronizedMap来，它提供了好得多的并发性。多个读操作几乎总可以并发地执行，同时进行的读和写操作通常也能并发地执行，而同时进行的写操作仍然可以不时地并发进行（相关的类也提供了类似的多个读线程的并发性，但是，只允许有一个活动的写线程）。Hashtable中采用的锁机制是一次锁住整个hash表，从而同一时刻只能由一个线程对其进行操作；而ConcurrentHashMap中则是一次锁住一个桶。ConcurrentHashMap默认将hash表分为16个桶，诸如get,put,remove等常用操作只锁当前需要用到的桶。这样，原来只能一个线程进入，现在却能同时有16个写线程执行，并发性能的提升是显而易见的。前面说到的16个线程指的是写线程，而读操作大部分时候都不需要用到锁。只有在size等操作时才需要锁住整个hash表。  
          在迭代方面，ConcurrentHashMap使用了一种不同的迭代方式。在这种迭代方式中，当iterator被创建后集合再发生改变就不再是抛出ConcurrentModificationException，取而代之的是在改变时new新的数据从而不影响原有的数据 ，iterator完成后再将头指针替换为新的数据 ，这样iterator线程可以使用原来老的数据，而写线程也可以并发的完成改变。
2. CopyOnWriteArrayList可以用于什么应用场景？  
           答：CopyOnWriteArrayList(免锁容器)的好处之一是当多个迭代器同时遍历和修改这个列表时，不会抛出ConcurrentModificationException。在CopyOnWriteArrayList中，写入将导致创建整个底层数组的副本，而源数组将保留在原地，使得复制的数组在被修改时，读取操作可以安全地执行。

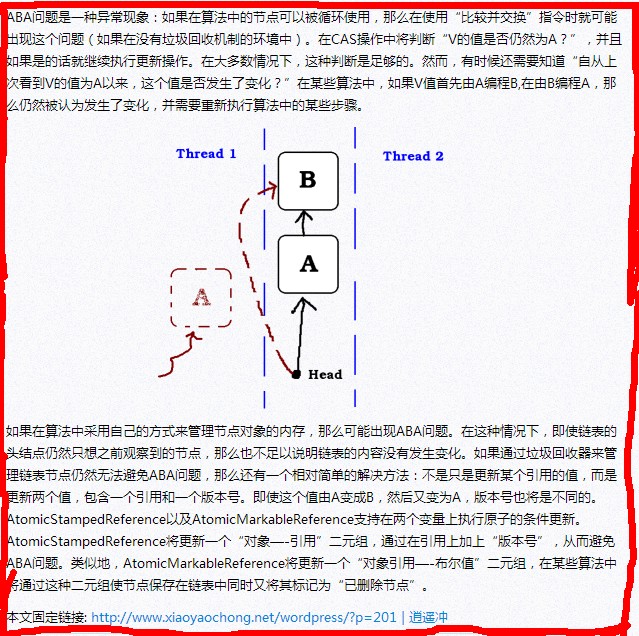
线程安全

1. 什么叫线程安全？servlet是线程安全吗?  
          答：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。 或者说:一个类或者程序所提供的接口对于线程来说是原子操作或者多个线程之间的切换不会导致该接口的执行结果存在二义性,也就是说我们不用考虑同步的问题。  
               servlet不是线程安全的，每个servlet都只被实例化一次，每个调用都是servlet的同一个实例，并且对类变量没有线程安全，数据量大的时候容易照成异常。
2. 同步有几种实现方法？  
        答：同步的实现方面有两种，分别是synchronized,wait与notify
3. volatile有什么用？能否用一句话说明下volatile的应用场景？  
      答：Volatile 变量具有 synchronized 的可见性特性，但是不具备原子特性。可以被看作是一种 “程度较轻的 synchronized”；与 synchronized 块相比，volatile 变量所需的编码较少，并且运行时开销也较少，但是它所能实现的功能也仅是 synchronized 的一部分。  
             您只能在有限的一些情形下使用 volatile 变量替代锁。要使 volatile 变量提供理想的线程安全，必须同时满足下面两个条件：  
                             A. 对变量的写操作不依赖于当前值。  
                             B. 该变量没有包含在具有其他变量的不变式中。
4. 请说明下java的内存模型及其工作流程。  
      答：Java把内存划分成两种：一种是栈内存，一种是堆内存。  
             栈内存：存放对象：函数中基本类型的变量和对象的引用变量、静态类方法 ；特点：栈有一个很重要的特殊性，就是存在栈中的数据可以共享。  
            堆内存：存放对象：用来存放由new创建的对象和数组；特点：在堆中分配的内存，由Java虚拟机的自动垃圾回收器来管理。  
             java 内存模型 ( java memory model )：根据Java Language Specification中的说明, jvm系统中存在一个主内存(Main Memory或Java Heap Memory)，Java中所有对象成员变量都储存在主存中，对于所有线程都是共享的。每条线程都有自己的工作内存(Working Memory)，工作内存中保存的是主存中某些对象成员变量的拷贝，线程对所有对象成员变量的操作都是在工作内存中进行，线程之间无法相互直接访问，变量传递均需要通过主存完成。  
                   (1) 获取对象监视器的锁(lock)  
                   (2) 清空工作内存数据, 从主存复制对象成员变量到当前工作内存, 即同步数据 (read and load)  
                   (3) 执行代码，改变共享变量值 (use and assign)  
                   (4) 将工作内存数据刷回主存 (store and write)  
                   (5) 释放对象监视器的锁 (unlock)
5. 为什么代码会重排序？

并发容器和框架

1. 如何让一段程序并发的执行，并最终汇总结果？  
    答：使用CyclicBarrier 和CountDownLatch都可以，使用CyclicBarrier 在多个关口处将多个线程执行结果汇总，CountDownLatch 在各线程执行完毕后向总线程汇报结果。
2. 如何合理的配置java线程池？如CPU密集型的任务，基本线程池应该配置多大？IO密集型的任务，基本线程池应该配置多大？用有界队列好还是无界队列好？任务非常多的时候，使用什么阻塞队列能获取最好的吞吐量？  
    答：1）配置线程池时CPU密集型任务可以少配置线程数，大概和机器的cpu核数相当，可以使得每个线程都在执行任务  
           2）IO密集型时，大部分线程都阻塞，故需要多配置线程数，2\*cpu核数  
           3）有界队列和无界队列的配置需区分业务场景，一般情况下配置有界队列，在一些可能会有爆发性增长的情况下使用无界队列。  
           4）任务非常多时，使用非阻塞队列使用CAS操作替代锁可以获得好的吞吐量。
3. 如何使用阻塞队列实现一个生产者和消费者模型？
4. 多读少写的场景应该使用哪个并发容器，为什么使用它？比如你做了一个搜索引擎，搜索引擎每次搜索前需要判断搜索关键词是否在黑名单里，黑名单每天更新一次。  
    答：1）CopyOnWriteArrayList这个容器适用于多读少写…  
           2）读写并不是在同一个对象上。在写时会大面积复制数组，所以写的性能差，在写完成后将读的引用改为执行写的对象

Java中的锁

1. 如何实现乐观锁（CAS）？如何避免ABA问题？  
    答：1）读取内存值的方式实现了乐观锁(比如：SVN系统)，方法：第一，比较内存值和期望值；第二，替换内存值为要替换值。  
           2）带参数版本来避免aba问题，在读取和替换的时候进行判定版本是否一致  
                      
2. 读写锁可以用于什么应用场景？  
    答： 读写锁可以用于 “多读少写” 的场景，读写锁支持多个读操作并发执行，写操作只能由一个线程来操作  
           ReadWriteLock对向数据结构相对不频繁地写入，但是有多个任务要经常读取这个数据结构的这类情况进行了优化。ReadWriteLock使得你可以同事有多个读取者，只要它们都不试图写入即可。如果写锁已经被其他任务持有，那么任何读取者都不能访问，直至这个写锁被释放为止。  
           ReadWriteLock 对程序心性能的提高受制于如下几个因素也还有其他等等的因素。  
             1）数据被读取的频率与被修改的频率相比较的结果。  
             2）读取和写入的时间  
             3）有多少线程竞争  
             4）是否在多处理机器上运行
3. 什么时候应该使用可重入锁？  
    答：重入锁指的是在某一个线程中可以多次获得同一把锁，在线程中多次操作有锁的方法。
4. 什么场景下可以使用volatile替换synchronized？  
    答： 只需要保证共享资源的可见性的时候可以使用volatile替代，synchronized保证可操作的原子性一致性和可见性。volatile适用于新值不依赖于就值的情形。  
         volatile是java提供的一种同步手段，只不过它是轻量级的同步，为什么这么说，因为volatile只能保证多线程的内存可见性，不能保证多线 程的执行有序性。而最彻底的同步要保证有序性和可见性，例如synchronized。任何被volatile修饰的变量，都不拷贝副本到工作内存，任何 修改都及时写在主存。因此对于Valatile修饰的变量的修改，所有线程马上就能看到，但是volatile不能保证对变量的修改是有序的。volatile存在的意义是，任何线程对某个变量的修改，都会马上被其他线程读取到，因为直接操作主存， 没有线程对工作内存和主存的同步。所以，volatile的使用场景是有限的，在有限的一些情形下可以使用 volatile 变量替代锁（synchronized）。  
          要使 volatile 变量提供理想的线程安全,必须同时满足下面两个条件:  
                  1)对 变量的写操作不依赖于当前值。  
                  2)该变量没有包含在具有其他变量的不变式中