# 网易电话面试

## 1 JUC

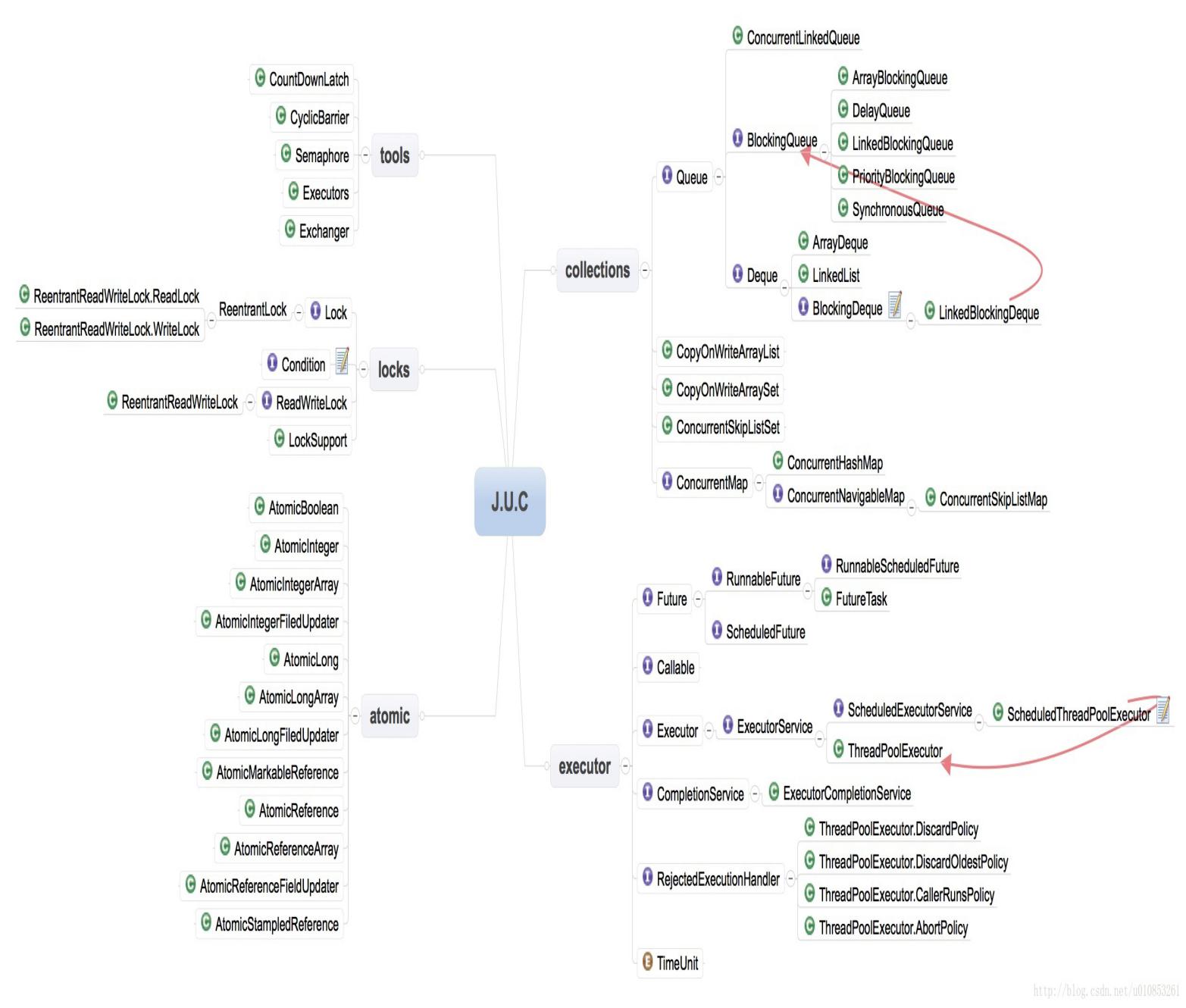
J.U.C 包下的并发类并发工具类

显示锁

原子变量类

并发集合

Executor线程执行器



## 2 多线程的东西同步机制

### 1.多线程有什么用？

1发挥多核CPU的优势，充分“压榨”CPU的性能

2防止阻塞。

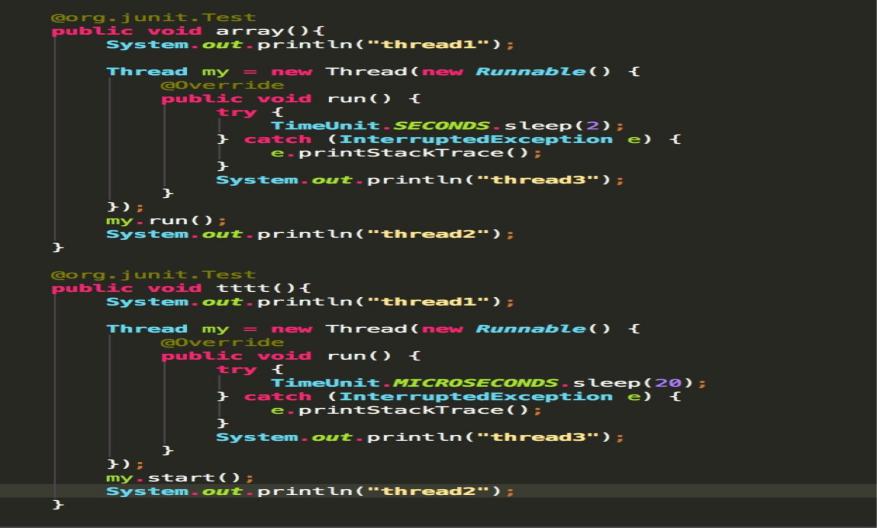
3便于建模，比如一个大的任务A，单线程完成要考虑很多，如果分成很多小任务，多线程来完成就简单很多了。

### 2.创建线程的方式

一般有三种：   
1）继承Thread类   
2）实现Runnable接口   
3）实现Callable接口   
一般来说我都是通过实现Runnable接口创建线程，也觉得这种方式好一些，毕竟Java一直提倡面向接口编程。

### 3. start()方法和run()方法的区别（unkonw）

start()是Thread里面的方法，在调用start()方法会立刻启动线程，然后会自动的调用run()方法，但是调用start()方法会立即返回，不会阻塞调用线程，从而真正实现多线程。   
1）start() 方法来启动线程，真正实现了多线程运行。这时无需等待run方法体代码执行完毕，可以直接继续执行下面的代码；通过调用Thread类的start()方法来启动一个线程， 这时此线程是处于就绪状态， 并没有运行。 然后通过此Thread类调用方法run()来完成其运行操作的， 这里方法run()称为线程体，它包含了要执行的这个线程的内容， Run方法运行结束， 此线程终止。然后CPU再调度其它线程。   
2）run（）方法当作普通方法的方式调用。程序还是要顺序执行，要等待run方法体执行完毕后，才可继续执行下面的代码； 程序中只有主线程——这一个线程， 其程序执行路径还是只有一条， 这样就没有达到多线程的目的。下图，第一个是132，第二个是123



### 4.Runnable接口和Callable接口的区别

1）Runnable接口里面的run()方法是没有返回值的，所以实现Runnable接口一般是执行不带返回值的任务。   
2）Callable接口里面的call()方法是有返回值的，返回值是一个泛型，和Future、FutureTask配合可以用来获取异步执行的结果。

有没有返回值也是这两个接口的主要区别，Callable接口的功能更加强大，使用Callable+Future/FutureTask可以获取多线程运行的结果，可以在等待时间太长没获取到需要的数据的情况下取消该线程的任务，非常有用。

### 5. CyclicBarrier和CountDownLatch的区别

这两个类都是J.U.C 并发包下面非常有用的类：   
1）CountDownLatch(闭锁)相当于一扇门：在闭锁到达结束状态之前（闭锁的结束状态也就是闭锁的计数器减到零），这扇门一直是关闭的，并且没有任何线程可以通过；当闭锁到达结束状态时，这扇门会打开并允许所有的等待线程(在闭锁上调用了await()方法的线程)通过。并且当闭锁到达结束状态后不可逆转，这扇门会一直保持打开的状态。应用场景：确保某些活动直到其他活动都结束了才继续执行。

2）CyclicBarrier 栏珊，它允许一组线程互相等待，直到都到达某个公共屏障点 (common barrier point)。在涉及一组固定大小的线程的程序中，这些线程必须不时地互相等待，此时 CyclicBarrier 很有用。因为该 barrier 在释放等待线程后可以重用，所以称它为循环 的 barrier。CyclicBarrier就象它名字的意思一样，可看成是个障碍， 所有的线程必须到齐后才能一起通过这个障碍。

### 6.volatile关键字的作用

简单点说就是提供了可视性机制。其实在理解volatile关键字的作用的前提是要理解Java内存模型，关于Java内存模型了，可以参见第31点，volatile关键字的作用主要有两个：   
（1）可视性：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）防止重排序：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。

volatile的一个重要作用就是和CAS结合，保证了原子性与可视性，详细的可以参见java.util.concurrent.atomic包下的类，比如AtomicInteger。

### 7.什么是线程安全

就我的理解来说：一段代码在单线程和多线程并发下永远都能获得一样的结果，那么就是线程安全的。

有值得一提的地方，就是线程安全也是有几个级别的：   
（1）不可变(对象)   
像String、Integer、Long这些，都是final类型的类，还有final类型的简单变量。任何一个线程都改变不了它们的值，要改变除非新创建一个，因此这些不可变对象不需要任何同步手段就可以直接在多线程环境下使用。   
（2）绝对线程安全   
不管运行时环境如何，调用者都不需要额外的同步措施。要做到这一点通常需要付出许多额外的代价，Java中标注自己是线程安全的类，实际上绝大多数都不是线程安全的，不过绝对线程安全的类，Java中也有，比方说CopyOnWriteArrayList、CopyOnWriteArraySet   
（3）相对线程安全   
相对线程安全也就是我们通常意义上所说的线程安全，像Vector这种，add、remove方法都是原子操作，不会被打断，但也仅限于此，如果有个线程在遍历某个Vector、有个线程同时在add这个Vector，99%的情况下都会出ConcurrentModificationException，也就是fail-fast机制。   
（4）线程非安全   
这个就没什么好说的了，ArrayList、LinkedList、HashMap等都是线程非安全的类，没有加任何保证线程安全的措施。

### 8.一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，那么这个对象锁会被立即释放

### 9.如何在两个线程之间共享数据

通过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，这里需要注意的是要保证多线程环境下的数据安全性。比方说阻塞队列BlockingQueue就是为线程之间共享数据而设计的，

### 10.sleep方法和wait方法有什么区别

相同点：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锁，sleep方法不会放弃这个对象的锁；wait方法会放弃这个对象的锁。

### 11.生产者消费者模型的作用是什么

1）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。   
2）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。

### 12.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。

### 13.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。

### 14.wait()方法和notify()/notifyAll()方法在放弃对象锁时有什么区别

wait()方法立即释放对象锁，notify()/notifyAll()方法则会等待线程剩余代码执行完毕才会放弃对象的锁。

### 15.为什么要使用线程池

1）避免频繁地创建和销毁线程，达到线程对象的重用。创建和销毁线程的开销还是很大的）  
2）使用线程池还可以根据项目灵活地控制并发的数目。

### 16.怎么检测一个线程是否持有对象监视器

有方法可以判断某个线程是否持有对象的锁：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。

### 17. synchronized和ReentrantLock（重入锁）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操作的是对象头中mark word。

### 18.ConcurrentHashMap的并发度是多少

看看源码就知道ConcurrentHashMap使用了分段锁，默认的并发度是16。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就不能同时有两条线程获取Hashtable中的数据。

### 19.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。

### 20. 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。

### 21. Java编程写一个会导致死锁的程序

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。

### 22.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](http://lib.csdn.net/base/operatingsystem" \o "操作系统知识库" \t "http://blog.csdn.net/u010853261/article/details/_blank)实现的，Java代码并没有办法直接接触到操作系统。

### 23.不可变对象对多线程有什么帮助

不可变对象保证了对象的内存可见性，而且不可变对象天生就是线程安全的。

### 24.什么是多线程的上下文切换

多线程的上下文切换是指CPU控制权由一个已经正在运行的线程切换到另外一个就绪并等待获取CPU执行权的线程的过程，这其中涉及到运行环境的保存与切换，存在时间开销。

### 25.如果你提交任务时，线程池队列已满，这时会发生什么

1）如果使用的是无界队列，比如LinkedBlockingQueue，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。比方说ArrayBlockingQueue的话，任务首先会被添加到ArrayBlockingQueue中，ArrayBlockingQueue满了，则会使用拒绝策略RejectedExecutionHandler处理满了的任务，默认是AbortPolicy。

### 26. 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。

### 27. Thread.sleep(0)的作用是什么

简单点说就是主动让出一次CPU：  
由于Java采用抢占式的线程调度[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/u010853261/article/details/_blank)，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。

### 28.什么是自旋

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。

### 29.什么是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

### 30.什么是CAS

CAS，全称为Compare and Swap，即比较-替换。假设有三个操作数：内存值V、旧的预期值A、要修改的值B，当且仅当预期值A和内存值V相同时，才会将内存值修改为B并返回true，否则什么都不做并返回false。当然CAS一定要volatile变量配合，这样才能保证每次拿到的变量是主内存中最新的那个值，否则旧的预期值A对某条线程来说，永远是一个不会变的值A，只要某次CAS操作失败，永远都不可能成功。

比如原子类：AtomicInteger 就是volatile+CAS实现的。

### 31.什么是乐观锁和悲观锁

其实这两种锁就是对线程安全的最悲观和最乐观的假设。   
（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，将比较-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。

### 32.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(DCL)单例模式的写法：线程安全

### 33.Semaphore有什么作用

常用作并发数控制器。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。

### 34. Hashtable的size()方法中明明只有一条语句”return count”，为什么还要做同步？

这是我之前的一个困惑，不知道大家有没有想过这个问题。某个方法中如果有多条语句，并且都在操作同一个类变量，那么在多线程环境下不加锁，势必会引发线程安全问题，这很好理解，但是size()方法明明只有一条语句，为什么还要加锁？

关于这个问题，在慢慢地工作、学习中，有了理解，主要原因有两点：  
（1）同一时间只能有一条线程执行固定类的同步方法，但是对于类的非同步方法，可以多条线程同时访问。所以，这样就有问题了，可能线程A在执行Hashtable的put方法添加数据，线程B则可以正常调用size()方法读取Hashtable中当前元素的个数，那读取到的值可能不是最新的，可能线程A添加了完了数据，但是没有对size++，线程B就已经读取size了，那么对于线程B来说读取到的size一定是不准确的。而给size()方法加了同步之后，意味着线程B调用size()方法只有在线程A调用put方法完毕之后才可以调用，这样就保证了线程安全性

（2）CPU执行代码，执行的不是Java代码，这点很关键，一定得记住。Java代码最终是被翻译成汇编代码执行的，汇编代码才是真正可以和硬件电路交互的代码。即使你看到Java代码只有一行，甚至你看到Java代码编译之后生成的字节码也只有一行，也不意味着对于底层来说这句语句的操作只有一个。一句”return count”假设被翻译成了三句汇编语句执行，完全可能执行完第一句，线程就切换了。

### 35. 线程类的构造方法、静态块是被哪个线程调用的？

这是一个非常刁钻和狡猾的问题。请记住：线程类的构造方法、静态块是被new这个线程类所在的线程所调用的，而run方法里面的代码才是被线程自身所调用的。

如果说上面的说法让你感到困惑，那么我举个例子，假设Thread2中new了Thread1，main函数中new了Thread2，那么：   
（1）Thread2的构造方法、静态块是main线程调用的，Thread2的run()方法是Thread2自己调用的

（2）Thread1的构造方法、静态块是Thread2调用的，Thread1的run()方法是Thread1自己调用的

### 36. 同步方法和同步块，哪个是更好的选择？

同步块，这意味着同步块之外的代码是异步执行的，这比同步整个方法更提升代码的效率。请知道一条原则：同步的范围越小越好。

借着这一条，我额外提一点，虽说同步的范围越少越好，但是在Java虚拟机中还是存在着一种叫做锁粗化的优化方法，这种方法就是把同步范围变大。这是有用的，比方说StringBuffer，它是一个线程安全的类，自然最常用的append()方法是一个同步方法，我们写代码的时候会反复append字符串，这意味着要进行反复的加锁->解锁，这对性能不利，因为这意味着Java虚拟机在这条线程上要反复地在内核态和用户态之间进行切换，因此Java虚拟机会将多次append方法调用的代码进行一个锁粗化的操作，将多次的append的操作扩展到append方法的头尾，变成一个大的同步块，这样就减少了加锁–>解锁的次数，有效地提升了代码执行的效率。

### 37. 什么是AQS？（\*）

简单说一下AQS，AQS全称为AbstractQueuedSychronizer，翻译过来应该是抽象队列同步器。

如果说java.util.concurrent的基础是CAS的话，那么AQS就是整个Java并发包的核心了，ReentrantLock、CountDownLatch、Semaphore等等都用到了它。AQS实际上以双向队列的形式连接所有的Entry，比方说ReentrantLock，所有等待的线程都被放在一个Entry中并连成双向队列，前面一个线程使用ReentrantLock好了，则双向队列实际上的第一个Entry开始运行。

AQS定义了对双向队列所有的操作，而只开放了tryLock和tryRelease方法给开发者使用，开发者可以根据自己的实现重写tryLock和tryRelease方法，以实现自己的并发功能。

### 38. 高并发、任务执行时间短的业务怎样使用线程池？并发不高、任务执行时间长的业务怎样使用线程池？并发高、业务执行时间长的业务怎样使用线程池？

这是我在并发编程网上看到的一个问题，把这个问题放在最后一个，希望每个人都能看到并且思考一下，因为这个问题非常好、非常实际、非常专业。关于这个问题，个人看法是：   
（1）高并发、任务执行时间短的业务，线程池线程数可以设置为CPU核数+1，减少线程上下文的切换

（2）并发不高、任务执行时间长的业务要区分开看：   
　　a）假如是业务时间长集中在IO操作上，也就是IO密集型的任务，因为IO操作并不占用CPU，所以不要让所有的CPU闲下来，可以加大线程池中的线程数目，让CPU处理更多的业务

　　b）假如是业务时间长集中在计算操作上，也就是计算密集型任务，这个就没办法了，和（1）一样吧，线程池中的线程数设置得少一些，减少线程上下文的切换

（3）并发高、业务执行时间长，解决这种类型任务的关键不在于线程池而在于整体[架构](http://lib.csdn.net/base/architecture" \o "大型网站架构知识库" \t "http://blog.csdn.net/u010853261/article/details/_blank)的设计，看看这些业务里面某些数据是否能做缓存是第一步，增加服务器是第二步，至于线程池的设置，设置参考（2）。最后，业务执行时间长的问题，也可能需要分析一下，看看能不能使用中间件对任务进行拆分和解耦。

### 39 wait方法是否必须用在while循环体中

是，《Effective Java》第二版中文版第69条244页位置对这一点说了一页，我看着一知半解。我能理解的一点是：对于从wait中被notify的进程来说，它在被notify之后还需要重新检查是否符合执行条件，如果不符合，就必须再次被wait，如果符合才能往下执行。所以：wait方法应该使用循环模式来调用。按照上面的生产者和消费者问题来说：错误情况一：如果有两个生产者A和B，一个消费者C。当存储空间满了之后，生产者A和B都被wait，进入等待唤醒队列。当消费者C取走了一个数据后，如果调用了notifyAll（），注意，此处是调用notifyAll（），则生产者线程A和B都将被唤醒，如果此时A和B中的wait不在while循环中而是在if中，则A和B就不会再次判断是否符合执行条件，都将直接执行wait（）之后的程序，那么如果A放入了一个数据至存储空间，则此时存储空间已经满了；但是B还是会继续往存储空间里放数据，错误便产生了。错误情况二：如果有两个生产者A和B，一个消费者C。当存储空间满了之后，生产者A和B都被wait，进入等待唤醒队列。当消费者C取走了一个数据后，如果调用了notify（），则A和B中的一个将被唤醒，假设A被唤醒，则A向存储空间放入了一个数据，至此空间就满了。A执行了notify（）之后，如果唤醒了B，那么B不会再次判断是否符合执行条件，将直接执行wait（）之后的程序，这样就导致向已经满了数据存储区中再次放入数据。错误产生。

### 40 Runnable 和 Thread的区别

在程序开发中只要是多线程肯定永远以实现Runnable接口为主，因为实现Runnable接口相比继承Thread类有如下好处：

* 避免点继承的局限，一个类可以继承多个接口。
* 适合于资源的共享

以卖票程序为例，通过Thread类完成要卖出30张票。虽然现在程序中有三个线程，但是一共卖了10张票，也就是说使用Runnable实现多线程可以达到资源共享目的。

## 3 计算机网络

### 1 TCP的三次握手和四次挥手

建立TCP需要三次握手才能建立，而断开连接则需要四次握手。  
首先Client端发送连接请求报文，Server段接受连接后回复ACK报文，并为这次连接分配资源。Client端接收到ACK报文后也向Server段发生ACK报文，并分配资源，这样TCP连接就建立了。

**【注意】中断连接端可以是Client端，也可以是Server端。**

假设Client端发起中断连接请求，也就是发送FIN报文。Server端接到FIN报文后，意思是说"我Client端没有数据要发给你了"，但是如果你还有数据没有发送完成，则不必急着关闭Socket，可以继续发送数据。所以你先发送ACK，"告诉Client端，你的请求我收到了，但是我还没准备好，请继续你等我的消息"。这个时候Client端就进入FIN\_WAIT状态，继续等待Server端的FIN报文。当Server端确定数据已发送完成，则向Client端发送FIN报文，"告诉Client端，好了，我这边数据发完了，准备好关闭连接了"。Client端收到FIN报文后，"就知道可以关闭连接了，但是他还是不相信网络，怕Server端不知道要关闭，所以发送ACK后进入TIME\_WAIT状态，如果Server端没有收到ACK则可以重传。“，Server端收到ACK后，"就知道可以断开连接了"。Client端等待了2MSL后依然没有收到回复，则证明Server端已正常关闭，那好，我Client端也可以关闭连接了。Ok，TCP连接就这样关闭了！

**【问题1】为什么连接的时候是三次握手，关闭的时候却是四次握手？**  
答：因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步握手。

**【问题2】为什么TIME\_WAIT状态需要经过2MSL(最大报文段生存时间)才能返回到CLOSE状态？**

答：虽然按道理，四个报文都发送完毕，我们可以直接进入CLOSE状态了，但是我们必须假象网络是不可靠的，有可以最后一个ACK丢失。所以TIME\_WAIT状态就是用来重发可能丢失的ACK报文。

### 2 HTTP协议

常见状态代码、状态描述、说明：  
200 OK      //客户端请求成功  
400 Bad Request  //客户端请求有语法错误，不能被服务器所理解  
401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用   
403 Forbidden  //服务器收到请求，但是拒绝提供服务  
404 Not Found  //请求资源不存在，eg：输入了错误的URL  
500 Internal Server Error //服务器发生不可预期的错误  
503 Server Unavailable  //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

 http请求由三部分组成，分别是：请求行、消息报头、请求正文

### 3 假如服务端知道 客户端IP（假设全球IP） 问服务端能否向 客户端发消息。

答案是不能，客户端接受消息是通过从服务器拉数据的形式来获取数据而不是服务器向客户端取数据。客户端几乎也是不会监听端口接受服务器发来的消息。

## 4 Java基础

### 1 问gc处理机制是怎么的还举了个String的例子然后说系统怎么判断是不用了的

垃圾回收器的运行时间是不确定的，由JVM决定，在运行时是间歇执行的。虽然可以通过System.gc()来强制回收垃圾，但是这个命令下达后无法保证JVM会立即响应执行，但经验表明，下达命令后，会在短期内执行你的请求。JVM通常会感到内存紧缺时候去执行垃圾回收操作。  
 垃圾回收过于频繁会导致性能下降，过于稀疏会导致内存紧缺。这个JVM会将其控制到最好，不用程序员担心。但有些程序在短期会吃掉大量内存，而这些恐怖的对象很快使用结束了，这时候也许有必要强制下达一条垃圾回命令，这是很有必要的，以便有更多可用的物理内存。  
 垃圾回收器仅仅能做的是尽可能保证可用内存的使用效率，让可用内存得到高效的管理。程序员可以影响垃圾回收的执行，但不能控制。  
 总之，在Java语言中，判断一块内存空间是否符合垃圾收集器收集标准的标准只有两个：  
 1．给对象赋予了空值null，以下再没有调用过。  
 2．给对象赋予了新值，既重新分配了内存空间。

引用计数算法和可达性分析算法

引用计数算法：给对象中添加一个引用计数器，每当有一个地方引用它时，计数器值加１；当引用失效时，计数器值减１.任何时刻计数器值为０的对象就是不可能再被使用的。无法解决对象之间相互循环引用。

可达性分析算法的基本思路是：通过一系列的称为“GC Roots”的对象作为起始点，从这些节点考试向下探索，搜索所走过的路径称为“引用链”，当一个对象到GC Roots没有任何引用链相连（用图论的话来说，就是从GC Roots到这个对象不可达）时，则证明此对象是不可用的。

### 2 hashmap与hashtable的区别是什么

**Hashtable** 继承自 **Dictionary 而 HashMap**继承自**AbstractMap**

****Hashtable的put方法（线程安全的）****

注意1 方法是同步的  
注意2 方法不允许value==null  
注意3 方法调用了key的hashCode方法，如果key==null,会抛出空指针异常

**HashMap的put方法（线程非安全的）**

注意1 方法是非同步的  
注意2 方法允许key==null  
注意3 方法并没有对value进行任何调用，所以允许为null

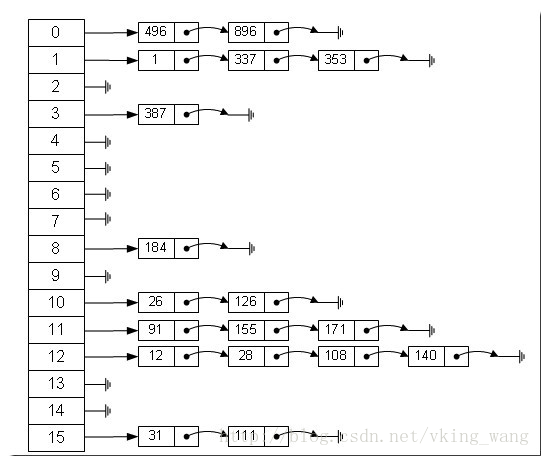
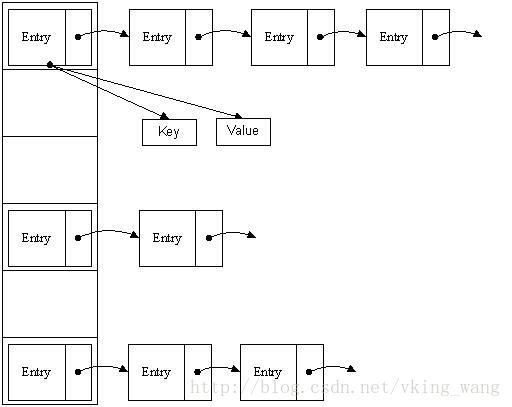
**补充：**   
Hashtable 有一个 contains方法，容易引起误会，所以在HashMap里面已经去掉了  
当然，2个类都用containsKey和containsValue方法。

HashMap是Hashtable的轻量级实现（非线程安全的实现），他们都完成了Map接口，

**主要区别在于HashMap允许空（null）键值（key）,由于非线程安全，效率上可能高于Hashtable。**

### 3 HashMap的原理

HashMap的数据结构就是数组加链表（有图就是拉链法实现）



**解决hash冲突的办法**

1. 开放定址法（线性探测再散列，二次探测再散列，伪随机探测再散列）
2. 再哈希法
3. **链地址法**
4. 建立一个公共溢出区

### 4 TreeMap的实现 TreeSet、红黑树

TreeMap 的实现就是红黑树数据结构，也就说是一棵自平衡的排序二叉树，这样就可以保证当需要快速检索指定节点。

TreeSet 和 TreeMap 的关系TreeSet 的 ① 号、② 号构造器的都是新建一个 TreeMap 作为实际存储 Set 元素的容器，而另外 2 个构造器则分别依赖于 ① 号和 ② 号构造器，由此可见，TreeSet 底层实际使用的存储容器就是 TreeMap。

**Java 实现的红黑树**

上面的性质 3 中指定红黑树的每个叶子节点都是空节点，而且并叶子节点都是黑色。但 Java 实现的红黑树将使用 null 来代表空节点，因此遍历红黑树时将看不到黑色的叶子节点，反而看到每个叶子节点都是红色的。

* 性质 1：每个节点要么是红色，要么是黑色。
* 性质 2：根节点永远是黑色的。
* 性质 3：所有的叶节点都是空节点（即 null），并且是黑色的。
* 性质 4：每个红色节点的两个子节点都是黑色。（从每个叶子到根的路径上不会有两个连续的红色节点）
* 性质 5：从任一节点到其子树中每个叶子节点的路径都包含相同数量的黑色节点。

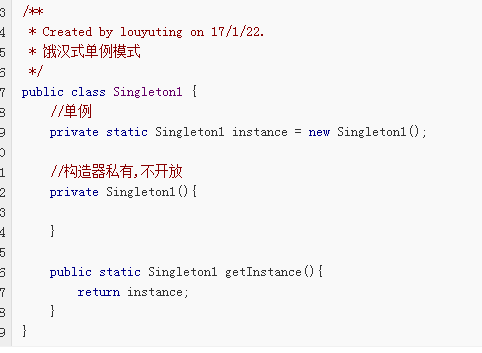
### 5 **[Java中List、Set、Map区别](http://blog.csdn.net/bug_moving/article/details/52667199)**

List,Set都是继承自Collection接口;

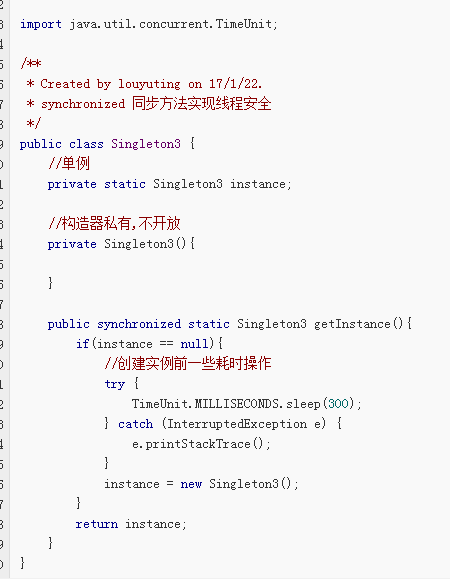
List特点：元素有放入顺序，元素可重复;   
Set特点：元素无放入顺序，元素不可重复（注意：元素虽然无放入顺序，但是元素在set中的位置是有该元素的HashCode决定的，其位置其实是固定的）;   
Map特点：元素按键值对存储，无放入顺序

### 6 单例的线程安全的程序

**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，所以肯定是线程安全的



线程安全的单例-synchronized方法



### 7 Object类有哪些方法

protected Object clone()创建并返回此对象的一个副本。   
boolean equals(Object obj)指示其他某个对象是否与此对象“相等”。   
protected void finalize()当垃圾回收器确定不存在对该对象的更多引用时，由对象的垃圾回收器调用此方法。   
Class<?> getClass()返回此 Object 的运行时类。   
int hashCode()返回该对象的哈希码值。   
void notify()唤醒在此对象监视器上等待的单个线程。   
void notifyAll()唤醒在此对象监视器上等待的所有线程。   
String toString()返回该对象的字符串表示。   
void wait()在其他线程调用此对象的 notify() 方法或 notifyAll() 方法前，导致当前线程等待。   
void wait(long timeout)在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者超过指定的时间量前，导致当前线程等待。   
void wait(long timeout, int nanos)在其他线程调用此对象的 notify() 方法或 notifyAll() 方法，或者其他某个线程中断当前线程，或者已超过某个实际时间量前，导致当前线程等待。

### 8 JVM堆栈

Java把内存划分成两种：一种是栈内存，一种是堆内存。  
　　在函数中定义的一些基本类型的变量和对象的引用变量都在函数的栈内存中分配。当在一段代码块定义一个变量时，Java就在栈中为这个变量分配内存空间，当超过变量的作用域后，Java会自动释放掉为该变量所分配的内存空间，该内存空间可以立即被另作他用。  
　　堆内存用来存放由new创建的对象和数组。在堆中分配的内存，由Java虚拟机的自动垃圾回收器来管理。  
　　在堆中产生了一个数组或对象后，还可以在栈中定义一个特殊的变量，让栈中这个变量的取值等于数组或对象在堆内存中的首地址，栈中的这个变量就成了数组或对象的引用变量。  
　　引用变量就相当于是为数组或对象起的一个名称，以后就可以在程序中使用栈中的引用变量来访问堆中的数组或对象。  
　　具体的说：栈与堆都是Java用来在Ram中存放数据的地方。与C++不同，Java自动管理栈和堆，程序员不能直接地设置栈或堆。  
　　Java的堆是一个运行时数据区,类的(对象从中分配空间。这些对象通过new、newarray、anewarray和 multianewarray等指令建立，它们不需要程序代码来显式的释放。堆是由垃圾回收来负责的，堆的优势是可以动态地分配内存大小，生存期也不必事先告诉编译器，因为它是在运行时动态分配内存的，Java的垃圾收集器会自动收走这些不再使用的数据。但缺点是，由于要在运行时动态分配内存，存取速度较慢。  
　　栈的优势是，存取速度比堆要快，仅次于寄存器，栈数据可以共享。但缺点是，存在栈中的数据大小与生存期必须是确定的，缺乏灵活性。栈中主要存放一些基本类型的变量（,int, short, long, byte, float, double, boolean, char）和对象句柄。  
　　栈有一个很重要的特殊性，就是存在栈中的数据可以共享。假设我们同时定义：  
　　int a = 3;  
　　int b = 3；  
　　编译器先处理int a = 3；首先它会在栈中创建一个变量为a的引用，然后查找栈中是否有3这个值，如果没找到，就将3存放进来，然后将a指向3。接着处理int b = 3；在创建完b的引用变量后，因为在栈中已经有3这个值，便将b直接指向3。这样，就出现了a与b同时均指向3的情况。这时，如果再令a=4；那么编译器会重新搜索栈中是否有4值，如果没有，则将4存放进来，并令a指向4；如果已经有了，则直接将a指向这个地址。因此a值的改变不会影响到b的值。要注意这种数据的共享与两个对象的引用同时指向一个对象的这种共享是不同的，因为这种情况a的修改并不会影响到b, 它是由编译器完成的，它有利于节省空间。而一个对象引用变量修改了这个对象的内部状态，会影响到另一个对象引用变量。

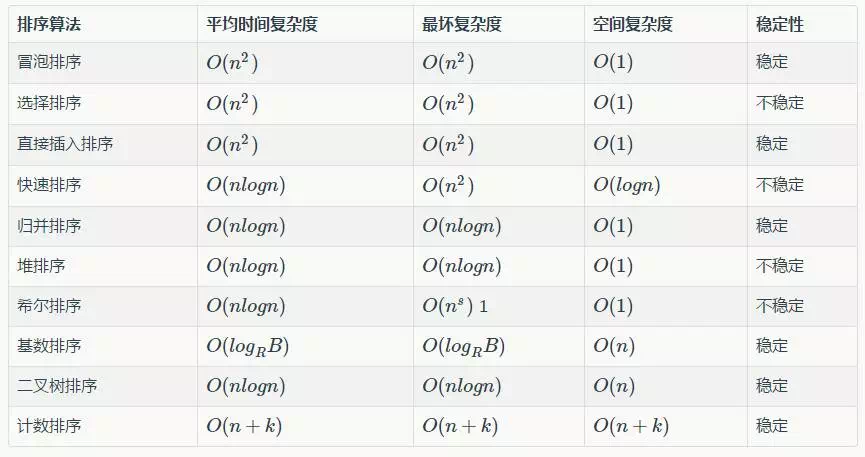
### 9 介绍一下Spring的IOC和AOP

IOC:控制反转也叫依赖注入。利用了工厂模式将对象交给容器管理，你只需要在spring配置文件总配置相应的bean，以及设置相关的属性，让spring容器来生成类的实例对象以及管理对象。在spring容器启动的时候，spring会把你在配置文件中配置的bean都初始化好，然后在你需要调用的时候，就把它已经初始化好的那些bean分配给你需要调用这些bean的类（假设这个类名是A），分配的方法就是调用A的setter方法来注入，而不需要你在A里面new这些bean了。  
注意：面试的时候，如果有条件，画图，这样更加显得你懂了  
AOP:面向切面编程。（Aspect-Oriented Programming）  
AOP可以说是对OOP的补充和完善。OOP引入封装、继承和多态性等概念来建立一种对象层次结构，用以模拟公共行为的一个集合。当我们需要为分散的对象引入公共行为的时候，OOP则显得无能为力。也就是说，OOP允许你定义从上到下的关系，但并不适合定义从左到右的关系。例如日志功能。日志代码往往水平地散布在所有对象层次中，而与它所散布到的对象的核心功能毫无关系。在OOP设计中，它导致了大量代码的重复，而不利于各个模块的重用。  
将程序中的交叉业务逻辑（比如安全，日志，事务等），封装成一个切面，然后注入到目标对象（具体业务逻辑）中去。  
实现AOP的技术，主要分为两大类：一是采用动态代理技术，利用截取消息的方式，对该消息进行装饰，以取代原有对象行为的执行；二是采用静态织入的方式，引入特定的语法创建“方面”，从而使得编译器可以在编译期间织入有关“方面”的代码  
简单点解释，比方说你想在你的biz层所有类中都加上一个打印‘你好’的功能这你经可以用aop思想来做，你先写个类写个方法，方法经实现打印‘你好’让后你Ioc这个类 ref＝“biz.\*”让每个类都注入。

利用java反射来管理Bean

DI 也大致的说了一下几个实现方式 1是基于接口的，2基于构造方法的，3是基于seter方法的

### 10 写一个快速排序



快排就是先一个定位函数，定位函数是找到一个数（可以是随机，一般用第一个数），然后把比这个数小的都放在左边，把比这个数大的都放在右边，然后快排递归调用这个定位函数即可。

### 11 SpringMVC的大概转发机制

SpingMVC的架构其实也很简单，它通过一个共通的入门DispatcherServlet来接收所有的request，接下来根据request要求的页面通过handler转送给Controller处理，处理结果返回View给用户，因此对SpringMVC来说，其核心为DispatcherServlet。  
DispatcherServlet作为一扇门，我们要使用这个门，首先得安装上这个门，安装在哪呢？没错，就安装在web.xml这个大屋子，象其他servlet一样，描述servlet的名字和类，再给他一个mapping描述对应映射的网页。

请求重定向与请求转发的比较，HttpServletResponse.sendRedirect方法和RequestDispatcher.forward方法都可以让浏览器获得另外一个URL所指向的资源，但两者的内部运行机制有很大的区别。

1.RequestDispatcher.forward方法只能将请求转发给同一个Web应用中的组件，HttpServletResponse.sendRedirect不仅可以重定向到当前应用程序的其他资源，还可以重定向到同一个站点上的其他应用程序的资源，甚至是使用绝对URL重定向到其他站点的资源。对于sendRedirect如果传递的URL以”/”开头，他是相对于整个Web站点的根目录；对于forward如果传递的URL以”/”开头，它是相对于当前Web应用程序的根目录。

2.sendRedirect对浏览器做出的响应是重新发出对另外一个URL的访问请求，forward在服务器端内部将请求转发给另外一个资源，浏览器只知道发出了请求并得到了响应结果

3.forward的调用者与被调用者之间共享相同的request对象和response对象，他们属于同一个访问请求和响应过程；sendRedirect的调用者与被调用者使用各自的request和response对象，属于两个独立的访问请求和响应过程

重定向和转发有一个重要的不同：当使用转发时，JSP容器将使用一个内部的方法来调用目标页面，新的页面继续处理同一个请求，而浏览器将不会知道这个过程。 与之相反，重定向方式的含义是第一个页面通知浏览器发送一个新的页面请求。因为，当你使用重定向时，浏览器中所显示的URL会变成新页面的URL, 而当使用转发时，该URL会保持不变。重定向的速度比转发慢，因为浏览器还得发出一个新的请求。同时，由于重定向方式产生了一个新的请求，所以经过一次重定向后，request内的对象将无法使用。

不要仅仅为了把变量传到下一个页面而使用session作用域，那会无故增大变量的作用域，转发也许可以帮助你解决这个问题。  
重定向：以前的request中存放的变量全部失效，并进入一个新的request作用域。  
转发：以前的request中存放的变量不会失效，就像把两个页面拼到了一起。

### 12 Linux命令

pwd   cd   ls  clear  cat   tail   grep  chmod  cp   mv  rm   mkdir  echo   tar

### 13 java的类加载机制

Class文件由类装载器装载后，在JVM中将形成一份描述Class结构的元信息对象，通过该元信息对象可以获知Class的结构信息：如构造函数，属性和方法等，Java允许用户借由这个Class相关的元信息对象间接调用Class对象的功能。

虚拟机把描述类的数据从class文件加载到内存，并对数据进行校验，转换解析和初始化，最终形成可以被虚拟机直接使用的Java类型，这就是虚拟机的类加载机制。

(1)装载：查找和导入Class文件；

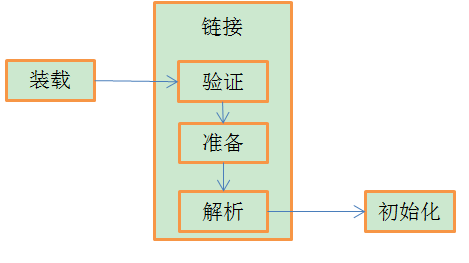
(2)链接：把类的二进制数据合并到JRE中；

(a)校验：检查载入Class文件数据的正确性；

(b)准备：给类的静态变量分配存储空间；

(c)解析：将符号引用转成直接引用；

(3)初始化：对类的静态变量，静态代码块执行初始化操作



类什么时候才被初始化：

1）创建类的实例，也就是new一个对象

2）访问某个类或接口的静态变量，或者对该静态变量赋值

3）调用类的静态方法

4）反射（Class.forName("com.lyj.load")）

5）初始化一个类的子类（会首先初始化子类的父类）

6）JVM启动时标明的启动类，即文件名和类名相同的那个类

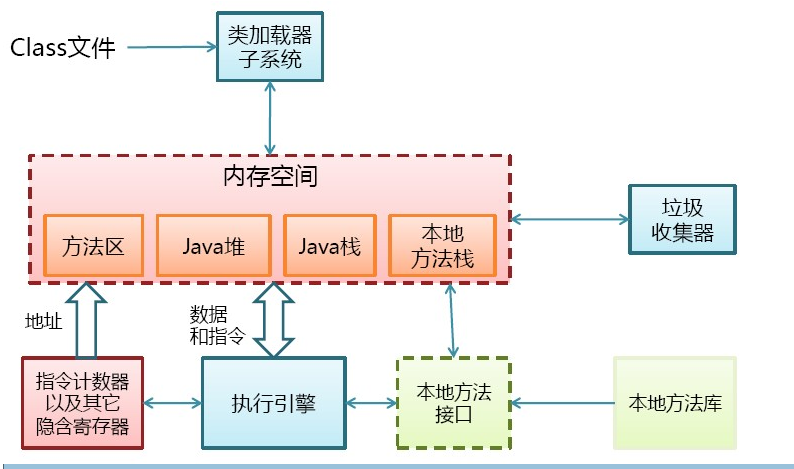
只有这6中情况才会导致类的类的初始化。

类的初始化步骤：

1）如果这个类还没有被加载和链接，那先进行加载和链接

2）假如这个类存在直接父类，并且这个类还没有被初始化（注意：在一个类加载器中，类只能初始化一次），那就初始化直接的父类（不适用于接口）

3)加入类中存在初始化语句（如static变量和static块），那就依次执行这些初始化语句。



### 14 事务的隔离级别

声明式事务：（注解，XML）  
编程式事务：（代码中嵌入）  
隔离级别：  
1、Serializable：最严格的级别，事务串行执行，资源消耗最大；  
2、REPEATABLE READ：保证了一个事务不会修改已经由另一个事务读取但未提交（回滚）的数据。避免了“脏读取”和“不可重复读取”的情况，但是带来了更多的性能损失。  
3、READ COMMITTED:大多数主流数据库的默认事务等级，保证了一个事务不会读到另一个并行事务已修改但未提交的数据，避免了“脏读取”。该级别适用于大多数系统。  
4、Read Uncommitted：保证了读取过程中不会读取到非法数据。  
隔离级别在于处理多事务的并发问题。我们知道并行可以提高数据库的吞吐量和效率，但是并不是所有的并发事务都可以并发运行，这需要查看数据库教材的可串行化条件判断了。  
事务的隔离级别的话，和数据库硬件相关

### java序列化的过程

1、什么是序列化和反序列化  
Serialization（序列化）是一种将对象以一连串的字节描述的过程；反序列化deserialization是一种将这些字节重建成一个对象的过程。

2、什么情况下需要序列化   
a）当你想把的内存中的对象保存到一个文件中或者[数据库](http://www.2cto.com/database/" \t "http://www.2cto.com/kf/201405/_blank)中时候；  
b）当你想用套接字在网络上传送对象的时候；  
c）当你想通过RMI传输对象的时候；

3、如何实现序列化

将需要序列化的类实现Serializable接口就可以了，Serializable接口中没有任何方法，可以理解为一个标记，即表明这个类可以序列化。

4、序列化和反序列化例子

如果我们想要序列化一个对象，首先要创建某些OutputStream(如FileOutputStream、ByteArrayOutputStream等)，然后将这些OutputStream封装在一个ObjectOutputStream中。这时候，只需要调用writeObject()方法就可以将对象序列化，并将其发送给OutputStream（记住：对象的序列化是基于字节的，不能使用Reader和Writer等基于字符的层次结构）。而反序列的过程（即将一个序列还原成为一个对象），需要将一个InputStream(如FileInputstream、ByteArrayInputStream等)封装在ObjectInputStream内，然后调用readObject()即可。会在此项目的工作空间生成一个 my.out文件。序列化后的内容稍后补齐，先看反序列化后输出如下：

name=SheepMu  
age=24

**序列化会忽略静态变量，即序列化不保存静态变量的状态**。静态成员属于类级别的，所以不能序列化。即 序列化的是对象的状态不是类的状态。这里的不能序列化的意思，是序列化信息中不包含这个静态成员域。最上面添加了static后之所以还是输出24是因为该值是JVM加载该类时分配的值。

总结:

a）当一个父类实现序列化，子类自动实现序列化，不需要显式实现Serializable接口；

b）当一个对象的实例变量引用其他对象，序列化该对象时也把引用对象进行序列化；

c） static,transient后的变量不能被序列化；

什么时候使用序列化：  
一：对象序列化可以实现分布式对象。主要应用例如：RMI要利用对象序列化运行远程主机上的服务，就像在本地机上运行对象时一样。  
二：java对象序列化不仅保留一个对象的数据，而且递归保存对象引用的每个对象的数据。可以将整个对象层次写入字节流中，可以保存在文件中或在网络连接上传递。利用对象序列化可以进行对象的"深复制"，即复制对象本身及引用的对象本身。序列化一个对象可能得到整个对象序列。

## 5 高并发分布式问题

### 1 1个亿的数据要我选出最大的10个数问时间空间最省的情况用什么算法,然后时间复杂度是多少

由于(1)输入的大量数据；(2)只要前K个，对整个输入数据的保存和排序是相当的不可取的。可以利用数据结构的最小堆来处理该问题。

最小堆如图所示，对于每个非叶子节点的数值，一定不大于孩子节点的数值。这样可用含有K个节点的最小堆来保存K个目前的最大值(当然根节点是其中的最小数值)。每次有数据输入的时候可以先与根节点比较。若不大于根节点，则舍弃；否则用新数值替换根节点数值。并进行最小堆的调整。

因为要前K个嘛，如果是最大堆，只能知道最上面的是最大的，而无法判断最小的节点具体在哪儿，就无法一下子知道下一个数是不是前K大 ，如果是最小堆，上面的元素就是目前前K的，如果比它大就说明这个不是前K大，换掉重新调整最小堆就行，如果比目前前K还小，直接剔除就行。

我只回答了会考虑到两个问题，一是数据存储问题（比如一个服务器挂了，怎么保证数据不受损失），另外一个是大量用户访问时候，服务器承受问题。具体怎么解决，用什么访问，我没有给出相关的解决方案，读到此文的希望查一下

## 6 数据库

### 1 sql复习

内链接

两张表如果有外键关系可以使用内链接，因为通过内链接，每一条只能返回单条记录

只要在关联表中查询出一条记录就可以使用内链接

外链接

只有部分记录可以从关联表查询到，主查询表想要显示所有记录，只能和关联表通过外链接

左外链接 右外链接

left join left左边是主查询表，left右边是关联查询表

right join right右边是主查询表，right左边是关联查询表

子查询

通过关联表的主键关联查询绝对没有问题，也可以不通过关联表的主键查询（但是子查询关联查询的结果集只能有一条记录）

在朱表上作子查询，主表记录会全部显示

嵌套表查询

将嵌套的结果集当成一个类，嵌套子查询中列不能重复的

嵌套表的层级有限制

### 2 事物的特性

事务四大特性(简称ACID)

1、原子性(Atomicity)：事务中的全部操作在[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/hao123huo/article/details/_blank)中是不可分割的，要么全部完成，要么均不执行。

2、一致性(Consistency)：几个并行执行的事务，其执行结果必须与按某一顺序串行执行的结果相一致。

3、隔离性(Isolation)：事务的执行不受其他事务的干扰，事务执行的中间结果对其他事务必须是透明的。

4、持久性(Durability)：对于任意已提交事务，系统必须保证该事务对数据库的改变不被丢失，即使数据库出现故障。

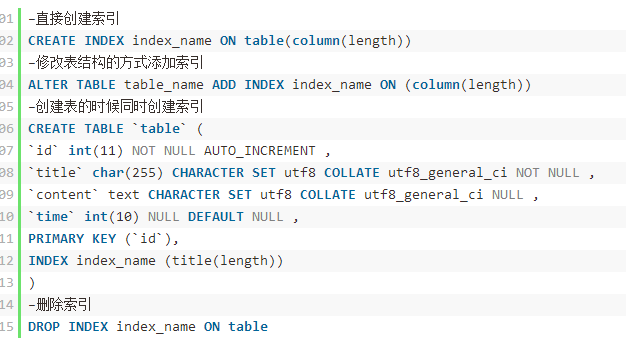
### 3 mysql的索引

索引分为聚簇索引和非聚簇索引两种，聚簇索引是按照数据存放的物理位置为顺序的，而非聚簇索引就不一样了；聚簇索引能提高多行检索的速度，而非聚簇索引对于单行的检索很快。

**MySQL索引的类型**

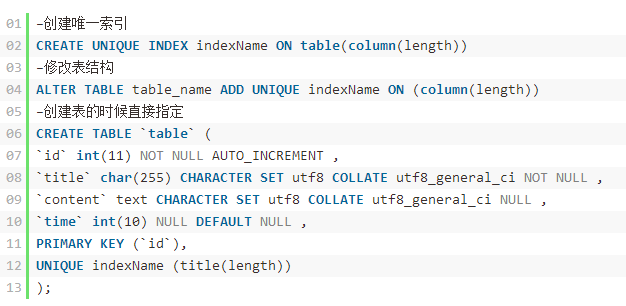
**1. 普通索引**

这是最基本的索引，它没有任何限制，比如上文中为title字段创建的索引就是一个普通索引，MyIASM中默认的BTREE类型的索引，也是我们大多数情况下用到的索引。



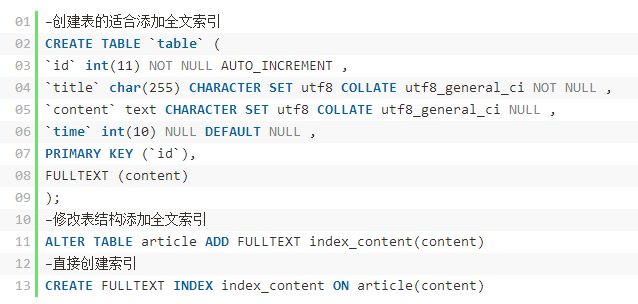
**2. 唯一索引**

与普通索引类似，不同的就是：索引列的值必须唯一，但允许有空值（注意和主键不同）。如果是组合索引，则列值的组合必须唯一，创建方法和普通索引类似。



**3. 全文索引（FULLTEXT）**

MySQL从3.23.23版开始支持全文索引和全文检索，FULLTEXT索引仅可用于 MyISAM 表；他们可以从CHAR、VARCHAR或TEXT列中作为CREATE TABLE语句的一部分被创建，或是随后使用ALTER TABLE 或CREATE INDEX被添加。////对于较大的数据集，将你的资料输入一个没有FULLTEXT索引的表中，然后创建索引，其速度比把资料输入现有FULLTEXT索引的速度更为快。不过切记对于大容量的数据表，生成全文索引是一个非常消耗时间非常消耗硬盘空间的做法。



**4. 单列索引、多列索引**

多个单列索引与单个多列索引的查询效果不同，因为执行查询时，MySQL只能使用一个索引，会从多个索引中选择一个限制最为严格的索引。

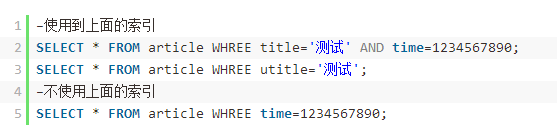
**5. 组合索引（最左前缀）**

平时用的SQL查询语句一般都有比较多的限制条件，所以为了进一步榨取MySQL的效率，就要考虑建立组合索引。例如上表中针对title和time建立一个组合索引：ALTER TABLE article ADD INDEX index\_titme\_time (title(50),time(10))。建立这样的组合索引，其实是相当于分别建立了下面两组组合索引：

–title,time

–title

为什么没有time这样的组合索引呢？这是因为MySQL组合索引“最左前缀”的结果。简单的理解就是只从最左面的开始组合。并不是只要包含这两列的查询都会用到该组合索引，如下面的几个SQL所示：



**MySQL索引的优化**

上面都在说使用索引的好处，但过多的使用索引将会造成滥用。因此索引也会有它的缺点：虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT、UPDATE和DELETE。因为更新表时，MySQL不仅要保存数据，还要保存一下索引文件。建立索引会占用磁盘空间的索引文件。一般情况这个问题不太严重，但如果你在一个大表上创建了多种组合索引，索引文件的会膨胀很快。索引只是提高效率的一个因素，如果你的MySQL有[大数据](http://lib.csdn.net/base/hadoop" \o "Hadoop知识库" \t "http://blog.csdn.net/xluren/article/details/_blank)量的表，就需要花时间研究建立最优秀的索引，或优化查询语句。下面是一些总结以及收藏的MySQL索引的注意事项和优化方法。

**1. 何时使用聚集索引或非聚集索引？**

|  |  |  |
| --- | --- | --- |
| **动作描述** | **使用聚集索引** | **使用非聚集索引** |
| 列经常被分组排序 | 使用 | 使用 |
| 返回某范围内的数据 | 使用 | 不使用 |
| 一个或极少不同值 | 不使用 | 不使用 |
| 小数目的不同值 | 使用 | 不使用 |
| 大数目的不同值 | 不使用 | 使用 |
| 频繁更新的列 | 不使用 | 使用 |
| 外键列 | 使用 | 使用 |
| 主键列 | 使用 | 使用 |
| 频繁修改索引列 | 不使用 | 使用 |

事实上，我们可以通过前面聚集索引和非聚集索引的定义的例子来理解上表。如：返回某范围内的数据一项。比如您的某个表有一个时间列，恰好您把聚合索引建立在了该列，这时您查询2004年1月1日至2004年10月1日之间的全部数据时，这个速度就将是很快的，因为您的这本字典正文是按日期进行排序的，聚类索引只需要找到要检索的所有数据中的开头和结尾数据即可；而不像非聚集索引，必须先查到目录中查到每一项数据对应的页码，然后再根据页码查到具体内容。其实这个具体用法我还不是很理解，只能等待后期的项目开发中慢慢学学了。

**2. 索引不会包含有NULL值的列**

只要列中包含有NULL值都将不会被包含在索引中，复合索引中只要有一列含有NULL值，那么这一列对于此复合索引就是无效的。所以我们在数据库设计时不要让字段的默认值为NULL。

**3. 使用短索引**

对串列进行索引，如果可能应该指定一个前缀长度。例如，如果有一个CHAR(255)的列，如果在前10个或20个字符内，多数值是惟一的，那么就不要对整个列进行索引。短索引不仅可以提高查询速度而且可以节省磁盘空间和I/O操作。

**4. 索引列排序**

MySQL查询只使用一个索引，因此如果where子句中已经使用了索引的话，那么order by中的列是不会使用索引的。因此数据库默认排序可以符合要求的情况下不要使用排序操作；尽量不要包含多个列的排序，如果需要最好给这些列创建复合索引。

**5. like语句操作**

一般情况下不鼓励使用like操作，如果非使用不可，如何使用也是一个问题。like “%aaa%” 不会使用索引而like “aaa%”可以使用索引。

**6. 不要在列上进行运算**

例如：select \* from users where YEAR(adddate)<2007，将在每个行上进行运算，这将导致索引失效而进行全表扫描，因此我们可以改成：select \* from users where adddate<’2007-01-01′。关于这一点可以围观：[一个单引号引发的MYSQL性能损失。](http://www.zendstudio.net/archives/single-quotes-or-no-single-quotes-in-sql-query" \t "http://blog.csdn.net/xluren/article/details/_blank)

很明显，不使用单引号没有用上主索引，并进行了全表扫描，使用单引号就能使用上索引了。

最后总结一下，MySQL只对一下操作符才使用索引：<,<=,=,>,>=,between,in,以及某些时候的like(不以通配符%或\_开头的情形)。而理论上每张表里面最多可创建16个索引，不过除非是数据量真的很多，否则过多的使用索引也不是那么好玩的，比如我刚才针对text类型的字段创建索引的时候，系统差点就卡死了。

### 4 数据库事务的隔离级别

**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/fg2006/article/details/_blank)**事务的隔离级别有4个，由低到高依次为Read uncommitted、Read committed、Repeatable read、Serializable，这四个级别可以逐个解决脏读、不可重复读、幻读这几类问题。

√: 可能出现    ×: 不会出现

|  |  |  |  |
| --- | --- | --- | --- |
|  | 脏读 | 不可重复读 | 幻读 |
| Read uncommitted | √ | √ | √ |
| Read committed | × | √ | √ |
| Repeatable read | × | × | √ |
| Serializable | × | × | × |

### 5 数据库回滚

rollback 回滚的意思。 就是数据库里做修改后 （ update ,insert , delete）未commit 之前 使用rollback 可以恢复数据到修改之前。

### 6 explain命令 如何查看mysql使用索引来执行select语句

对mysql explain讲的比较清楚的

在 explain的帮助下，您就知道什么时候该给表添加索引，以使用索引来查找记录从而让select 运行更快。  
如果由于不恰当使用索引而引起一些问题的话，可以运行 analyze table来更新该表的统计信息，例如键的基数，它能帮您在优化方面做出更好的选择。

explain 返回了一行记录，它包括了 select语句中用到的各个表的信息。这些表在结果中按照mysql即将执行的查询中读取的顺序列出来。mysql用一次扫描多次连接（single- sweep,multi-join）的方法来解决连接。这意味着mysql从第一个表中读取一条记录，然后在第二个表中查找到对应的记录，然后在第三个表中查找，依次类推。当所有的表都扫描完了，它输出选择的字段并且回溯所有的表，直到找不到为止，因为有的表中可能有多条匹配的记录下一条记录将从该表读取，再从下一个表开始继续处理。  
在mysql version 4.1中，explain输出的结果格式改变了，使得它更适合例如 union语句、子查询以及派生表的结构。更令人注意的是，它新增了2个字段： id和 select\_type。当你使用早于mysql4.1的版本就看不到这些字段了。

7 数据库设计时，字段可以选择默认值或者null，为何选择默认值

null值在有些存储引擎中按最大字节数存储，占空间，null不能被索引，影响性能；

### 8 大规模的delete或insert操作是否会对引起表锁定，如何解决

分批次delete或insert

### 9 怎么在SQL层面做一些优化

sql层面优化：尽量不要用\*,in和not in等  
数据库层面优化：走索引，建分区表。

SQL优化基本就是一些常识性的东西，比如不能用SELELCT \*，少用DISTINCT、GROUP BY之类的命令。

SQL优化步骤

1.   使用pt-query-digest分析慢日志，拿到调用频率高并且执行时间长的语句。

2.   获取语句相关表的信息，以及explain等相关信息，一般收集如下信息

* l  执行计划：explain query;
* l  详细执行计划：explain extended query;show warnings;
* l  相关表信息：show create table test1;
* l  语句执行开销：show profile for query num;
* l  数据量：select count(0) from table;
* l  相关表的索引信息：show index for table;
* l  my.cnf查询相关参数如各类buffer等

3.   进行初步分析，诊断SQL语句，查看在SQL层面能否带来提升，一般应对手段如下

* 条件上拉、下推
* 子查询合并、展开
* 添加索引、优化索引
* distinct、group by、orderby尽量利用索引减少文件排序
* 子查询尽量转换为join，最好消除
* 尽量减少聚合函数，将复杂查询转换为多表链接
* 尽量优化等价谓词，尽可能利用索引
* join的一些优化，连接消除、去除中间表等

4.   SQL无优化空间考虑从业务逻辑层面入手。常用手段如下

* l  增加冗余字段
* l  拆分表（水平/竖直）
* l  使用分布式
* l  将大事务拆分为多个小事务
* l  精简数据库的设计，减少使用存储过程、触发器等

5.   对硬件进行优化，升级服务器硬件。

3、4步骤可与调整系统参数相结合，具体使用方法视具体业务而定。

### 10 **[MySQL性能优化——易实现的MySQL优化方案汇总](http://blog.csdn.net/zhangliangzi/article/details/52329355)**

1. **索引优化**

1、合理使用索引，在经常查询而不经常增删改操作的字段加索引，一个表上的索引不应该超过6个。

2、Order by与group by后应直接使用字段，而且字段应该是索引字段。

3、索引字段长度应较短而长度固定。

4、索引字段重复不能过多。

**5、Hash索引与BTree索引区别（MyISAM与InnoDB不支持Hash索引）**

(1)、BTree索引使用多路搜索树的**[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/zhangliangzi/article/details/_blank)**，可以减少定位的中间过程；综合效率较高，默认使用的索引。

(2)、Hash索引使用Hash**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/zhangliangzi/article/details/_blank)**构建索引；精确的等值查询一次定位，效率极高，但特别不适合范围查询；使用Hash的复合索引是把复合索引键共同计算hash值，故不能单独使用。

**6、会导致引擎放弃使用索引，改为进行全表的几种情况，都要在开发中尽量避免出现！**

(1)、where子句中使用like关键字时，前置百分号会导致索引失效（起始字符不确定都会失效）。如：select id from test where name like "%吉坤"。

(2)、where子句中使用is null或is not null时，因为null值会被自动从索引中排除，索引一般不会建立在有空值的列上。

(3)、where子句中使用or关键字时，or左右字段如果存在一个没有索引，有索引字段也会失效；而且即使都有索引，因为二者的索引存储顺序并不一致，效率还不如顺序全表扫描，这时引擎有可能放弃使用索引，所以要慎用or。

(4)、where子句中使用in或not in关键字时，会导致全表扫描，能使用exists或between and替代就不使用in。

(5)、where子句中使用!=操作符时，将放弃使用索引，因为范围不确定，使用索引效率不高，会被引擎自动改为全表扫描；

(6)、where子句中应尽量避免对索引字段操作（表达式操作或函数操作），比如select id from test where num/2 = 100应改为num = 200。

(7)、在使用复合索引时，查询时必须使用到索引的第一个字段，否则索引失效；并且应尽量让字段顺序与索引顺序一致。

(8)、查询时必须使用正确的数据类型。数据库包含了自动了类型转换，比如纯数字赋值给字符串字段时可以被自动转换，但如果查询时不加引号查询，会导致引擎忽略索引。

1. **表结构优化**
2. 设计符合第三范式的表结构。

2、尽量使用数字型字段，提高数据比对效率。

3、对定长、MD5哈希码、长度较短的字段使用char类型，提高效率；对边长而且可能较长字段使用varchar类型，节约内存。

4、适当的进行水平分割与垂直分割，比如当表列数过多时，就将一部分列移出到另一张表中。

**关于水平分割与垂直分割表详解：**

**水平分割表：**一种是当多个过程频繁访问数据表的不同行时，水平分割表，并消除新表中的冗余数据列；若个别过程要访问整个数据，则要用连接\*作，这也无妨分割表；典型案例是电信话单按月分割存放。另一种是当主要过程要重复访问部分行时，最好将被重复访问的这些行单独形成子集表（冗余储存），这在不考虑磁盘空间开销时显得十分重要；但在分割表以后，增加了维护难度，要用触发器立即更新、或存储过程或应用代码批量更新，这也会增加额外的磁盘I/O开销。

水平分割会给应用增加复杂度，它通常在查询时需要多个表名，查询所有数据需要union操作。在许多数据库应用中，这种复杂性会超过它带来的优点，因为只要索引关键字不大，则在索引用于查询时，表中增加两到三倍数据量，查询时也就增加读一个索引层的磁盘次数。

**垂直分割表（不破坏第三范式）**：一种是当多个过程频繁访问表的不同列时，可将表垂直分成几个表，减少磁盘I/O（每行的数据列少，每页存的数据行就多，相应占用的页就少），更新时不必考虑锁，没有冗余数据。缺点是要在插入或删除数据时要考虑数据的完整性，用存储过程维护。另一种是当主要过程反复访问部分列时，最好将这部分被频繁访问的列数据单独存为一个子集表（冗余储存），这在不考虑磁盘空间开销时显得十分重要；但这增加了重叠列的维护难度，要用触发器立即更新、或存储过程或应用代码批量更新，这也会增加额外的磁盘I/O开销。垂直分割表可以达到最大化利用Cache的目的。

垂直分割可以使得数据行变小(因为列少了，一行数据就变小)，一个数据页就能存放更多的数据，在查询时就会减少I/O 次数。其缺点是需要管理冗余列，查询所有数据需要join操作 。

1. **临时表优化——临时表常常用于排序或分组，所以Order By与Group By后的字段尽量使用索引**

临时表可以根据实际需求使用，但要尽力避免磁盘临时表的生成。

**1、常见的会产生内存临时表的情况**

1、UNION查询。  
2、子查询（所以我们一般用join代替子查询）。  
3、join查询中，如果order by 与 group by如果使用的不都是第一张表上的字段，就会产生临时表。  
4、order by中使用distinct函数。

**2、常见的会产生磁盘临时表的情况**

1、数据表中包含BLOB/TEXT列。  
2、Group by、distinct、union查询中包含超过512字节的列。

1. **其他优化**

1、不使用Select \*，只查询需要的字段。

2、在只查询一条字段时，limit 1。

3、避免大事务操作，提高并发能力。

4、在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送 DONE\_IN\_PROC 消息。

5、尽量少使用游标。

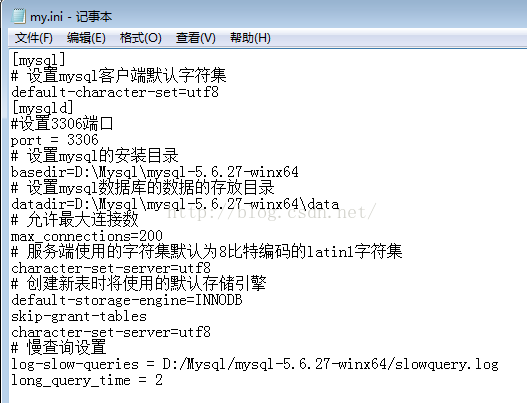
6、多去关注慢查询，总有我们提前考虑不到的问题，出现了就去解决它！

**附慢查询开启方式：**

在**[MySQL](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "http://blog.csdn.net/zhangliangzi/article/details/_blank)**安装目录下，找到my.ini配置文件，在mysqld下加上如下配置：

log-slow-queries = D:/Mysql/mysql-5.6.27-winx64/slowquery.log

long\_query\_time = 2



### 11 三范式

第一：列满足原子性，即求每一列都不允许再次拆分（列不可分）  
如：表中若有“地址"列。则地址还可以拆分为：国家、省份、城市等这些列，就说明地址这列还可拆分，则不满足第三范式  
第二：满足第一的基础上，除主键以外每一列都依赖于主键（不能有部分依赖）  
如：一张表是描述学员的；有学员编号，学校名称；其中学校名称和学员编号没有依赖关系。应把学校名称放在学校表中  
第三：满足第一第二的基础上，除主键以外的列都直接依赖主键。（不能有传递依赖）

如：城镇表依赖市表，市表依赖省表，则可以推断出城镇依赖省表；现在城镇表和省表之间就是间接相关

### 12 select for update

**MySQL SELECT ... FOR UPDATE 的Row Lock 与Table Lock**  
上面介绍过SELECT ... FOR UPDATE 的用法，不过锁定(Lock)的数据是判别就得要注意一下了。由于InnoDB 预设是Row-Level Lock，所以只有「明确」的指定主键，MySQL 才会执行Row lock (只锁住被选取的数据) ，否则MySQL 将会执行Table Lock (将整个数据表单给锁住)。

举个例子:  
假设有个表单products ，里面有id 跟name 二个栏位，id 是主键。  
例1: (明确指定主键，并且有此数据，row lock)

SELECT \* FROM products WHERE id='3' FOR UPDATE;

例2: (明确指定主键，若查无此数据，无lock)

SELECT \* FROM products WHERE id='-1' FOR UPDATE;

例2: (无主键，table lock)

SELECT \* FROM products WHERE name='Mouse' FOR UPDATE;

例3: (主键不明确，table lock)

SELECT \* FROM products WHERE id<>'3' FOR UPDATE;

例4: (主键不明确，table lock)

SELECT \* FROM products WHERE id LIKE '3' FOR UPDATE;

**乐观所和悲观锁策略**  
悲观锁：在读取数据时锁住那几行，其他对这几行的更新需要等到悲观锁结束时才能继续 。  
乐观所：读取数据时不锁，更新时检查是否数据已经被更新过，如果是则取消当前更新，一般在悲观锁的等待时间过长而不能接受时我们才会选择乐观锁。