**一、谈谈对Java的认识**  
1、基本介绍  
 Java是一种跨平台的面向对象的开发语言。具有 简单性，面向对象，分布式，可靠，安全，解释型、平台无关，高性能，多线程，动态性等特点。

2、基本认识

（1）、简单。Java丢弃了C++ 中很少使用的、很难理解的、令人迷惑的那些特性，如操作符重载、多继承、自动的强制类型转换。特别是，Java语言不使用指针，并提供了自动的垃圾回收机制，使得程序员不必为内存管理而担忧。

（2）、面向对象。封装、继承和多态三大特点； 只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口之间的实现机制（关键字为implements）。Java语言全面支持动态绑定。总之，Java语言是一个纯的面向对象程序设计语言。

（3）、分布式。Java语言支持网络应用的开发，在基本的Java应用编程接口中有一个网络应用编程接口（java.net），它提供了用于网络应用编程的类库，包括URL、URLConnection、Socket、 ServerSocket等。Java的RMI(远程方法激活)机制也是开发分布式应用的重要手段。

（4）、可靠（健壮）。Java的强类型机制、异常处理、垃圾的自动收集等是Java程序健壮性的重要保证。

（5）、安全。Java通常被用在网络环境中，Java提供了一个安全机制以防恶意代码的攻击。除了Java语言具有的许多安全特性以外，Java对通过网络下载的类具有一个安全防范机制（类ClassLoader），如分配不同的名字空间以防替代本地的同名类、字节代码检查，并提供安全管理机制（类SecurityManager）让Java应用设置安全哨兵。

（6）、解释型。Java程序在Java平台上被编译为字节码，然后可以在实现Java平台的任何系统中运行。在运行时，Java平台中的Java解释器对这些字节码进行解释执行，执行过程中需要的类在联接阶段被载入到运行环境中。

（7）、平台无关。是指JAVA写的应用程序不用修改就可以在不同的软硬件平台上运行。依靠JAVA虚拟机JVM来实现，他依附在具体的操作系统上，有一套自己的虚机器指令，在JVM上有一个JAVA解释器用来解释JAVA编译器编译后的程序。

（8）、高性能。与解释型的高级脚本语言相比，Java的确是高性能的。事实上，Java的运行速度随着编译器技术的发展越来越接近于C++。

（9）、多线程。**JAVA**多线程实现方式主要有三种：

* 继承Thread类



* 实现Runnable接口



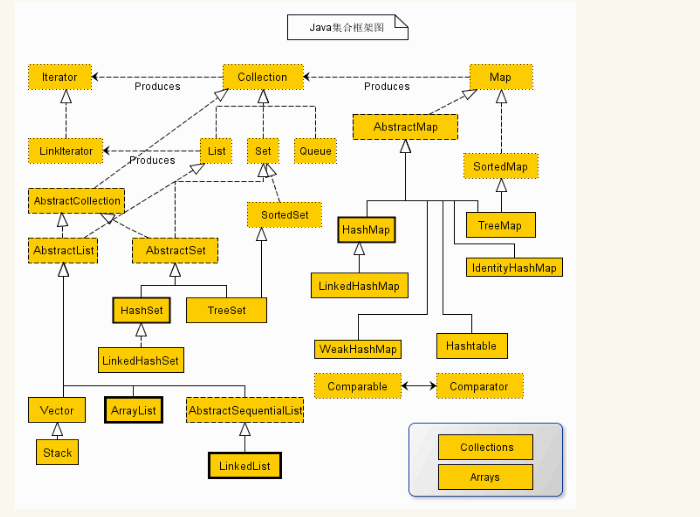
* 使用ExecutorService、Callable、Future实现有返回结果的多线程。

注：其中前两种方式线程执行完后都没有返回值，只有最后一种是带返回值的。

（10）、动态性。Java语言的设计目标之一是适应于动态变化的环境。Java程序需要的类能够动态地被载入到运行环境，也可以通过网络来载入所需要的类。这也有利于软件的升级。另外，Java中的类有一个运行时刻的表示，能进行运行时刻的类型检查。

**二、Java集合类解析**

详情链接：<http://www.cnblogs.com/leeplogs/p/5891861.html>



1、在使用Java的时候，我们都会遇到使用集合（[Collection](https://baike.baidu.com/item/Collection/80124)）的时候，但是[Java](https://baike.baidu.com/item/Java/85979) [API](https://baike.baidu.com/item/API/10154)提供了多种集合的实现。总的说来，[Java API](https://baike.baidu.com/item/Java%20API)中所用的集合类，都是实现了[Collection接口](https://baike.baidu.com/item/Collection%E6%8E%A5%E5%8F%A3)，他的一个类继承结构如下：

Collection<--List<--AbstractList<--[Vector](https://baike.baidu.com/item/Vector/3330482)

Collection<--List<--AbstractList <--[ArrayList](https://baike.baidu.com/item/ArrayList)

Collection<--List<--AbstractList <--AbstractSequentialList<--LinkedList

Collection<--Set<--AbstractSet<--[HashSet](https://baike.baidu.com/item/HashSet)

Collection<--Set<--AbstractSet<--HashSet<--LinkedHashSet

Collection<--Set<--SortedSet<--TreeSet

Collection<--Map<--AbstractMap<--HashMap<--LinkedHashMap

Collection<--Map<--AbstractMap<--WeakHashMap

Collection<--Map<--AbstractMap<--HashTable

Collection<--Map<--AbstractMap<--IdentityHashMap

Collection<--Map<--SortedMap<--TreeMap

注：

Vector : 基于Array的List，其实就是封装了Array所不具备的一些功能方便我们使用，它不可能走出Array的限制。性能也就不可能超越Array。所以，在可能的情况下，我们要多运用Array。

另外很重要的一点就是Vector“synchronized”的，这个也是Vector和ArrayList的唯一的区别。

[ArrayList](https://baike.baidu.com/item/ArrayList)：同Vector一样是一个基于数组实现的，但是不同的是ArrayList不是同步的，所以在性能上要比Vector优越一些。但是当运行到多线程环境中时，可需要自己在管理线程的同步问题。与Vector相比，如果不考虑线程安全因素，一般使用arrayList效率较高。

如果集合中的元素的数目大于目前集合数组的长度时，vector增长率为目前数组长度的100%，而arraylist增长率为目前数组长度的50%。如果在集合中使用数据量比较大的数据，用vector有一定的优势。

LinkedList：ArrayList是实现了基于动态数组的数据结构，LinkedList基于链表的数据结构。LinkedList也是非同步的。它每一个节点（Node）都包含两方面的内容：1.节点本身的数据（data）；2.下一个节点的信息（nextNode）。所以当对LinkedList做添加，删除动作的时候就不用像基于Array的List一样，必须进行大量的数据移动。只要更改nextNode的相关信息就可以实现了。这就是LinkedList的优势。

**2、Map集合**

实现类：HashMap、Hashtable（同步的）、LinkedHashMap和TreeMap

HashMap

HashMap是最常用的Map，它根据键的HashCode值存储数据，根据键可以直接获取它的值，具有很快的访问速度，遍历时，取得数据的顺序是完全随机的。因为键对象不可以重复，所以HashMap最多允许一条记录的键为Null，允许多条记录的值为Null，是非同步的

Hashtable

Hashtable与HashMap类似，是HashMap的线程安全版，它支持线程的同步，即任一时刻只有一个线程能写Hashtable，因此也导致了Hashtale在写入时会比较慢，效率较低。它不允许记录的key或者value为null。

LinkedHashMap

LinkedHashMap保存了记录的插入顺序，在用Iteraor遍历LinkedHashMap时，先得到的记录肯定是先插入的，在遍历的时候会比HashMap慢，有HashMap的全部特性。

TreeMap

TreeMap实现SortMap接口，能够把它保存的记录根据键排序，默认是按键值的升序排序（自然顺序），也可以指定排序的比较器，当用Iterator遍历TreeMap时，得到的记录是排过序的。不允许key值为空，非同步的；

ConcurrentHashMap

线程安全，并且锁分离。ConcurrentHashMap内部使用段(Segment)来表示这些不同的部分，每个段其实就是一个小的hash table，它们有自己的锁。只要多个修改操作发生在不同的段上，它们就可以并发进行。

**3、实现原理**

（1）、HashMap的实现原理

HashMap实际上是一个“链表散列”的数据结构，即数组和链表的结合体。

<http://zhangshixi.iteye.com/blog/672697>

（2）、TreeMap实现原理

TreeMap的实现是红黑树[算法](http://lib.csdn.net/base/datastructure)的实现。

<http://blog.csdn.net/a616413086/article/details/52586006>

**4、HashMap和HashSet的区别**

<http://www.importnew.com/6931.html>

**三、Java中多线程的内容--同步机制**

参考网址：<http://blog.csdn.net/u012891504/article/details/51332298>

1、多线程的线程同步机制实际上是靠锁的概念来控制的。

2、在Java程序运行时环境中，JVM需要对两类线程共享的数据进行协调：  
1）保存在堆中的实例变量  
2）保存在方法区中的类变量

这两类数据是被所有线程共享的。程序不需要协调保存在Java 栈当中的数据。因为这些数据是属于拥有该栈的线程所私有的。

3、在java虚拟机中，每个对象和类在逻辑上都是和一个监视器相关联的。

1）对于对象来说，相关联的监视器保护对象的实例变量；

2）对于类来说，监视器保护类的类变量；

为了实现监视器的排他性监视能力，java虚拟机为每一个对象和类都关联一个锁。代表任何时候只允许一个线程拥有的特权。线程访问实例变量

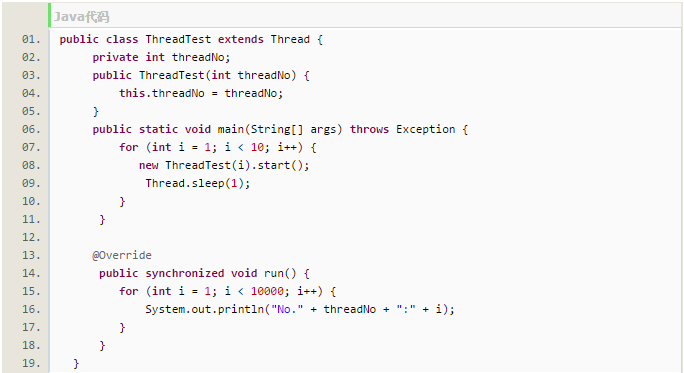
或者类变量不需锁。但是如果线程获取了锁，那么在它释放这个锁之前，就没有其他线程可以获取同样数据的锁了。（锁住一个对象就是获取对象相关联的监视器）

4、类锁实际上用对象锁来实现。

当虚拟机装载一个class文件的时候，它就会创建一个java.lang.Class类的实例。当锁住一个对象的时候，实际上锁住的是那个类的Class对象。  
一个线程可以多次对同一个对象上锁。对于每一个对象，java虚拟机维护一个加锁计数器，线程每获得一次该对象，计数器就加1，每释放一次，计数器就减 1，当计数器值为0时，锁就被完全释放了。java编程人员不需要自己动手加锁，对象锁是java虚拟机内部使用的。  
5、在java程序中，只需要使用synchronized块或者synchronized方法就可以标志一个监视区域。当每次进入一个监视区域时，java 虚拟机都会自动锁上对象或者类。  
重点：当一个有限的资源被多个线程共享的时候，为了保证对共享资源的互斥访问，我们一定要给他们排出一个先来后到：对象锁在这里起着非常重要的作用。多线程是如何处理共享资源的，以及保证他们对资源进行互斥访问所依赖的重要机制：对象锁。

**6、传统的同步实现方式以及这背后的原理**

（1）、Java多线程编程中，重要的关键字-->synchronized。“但是这么一个关键字，我也看不出来Java程序锁住了哪个对象阿？“ 示例如下：

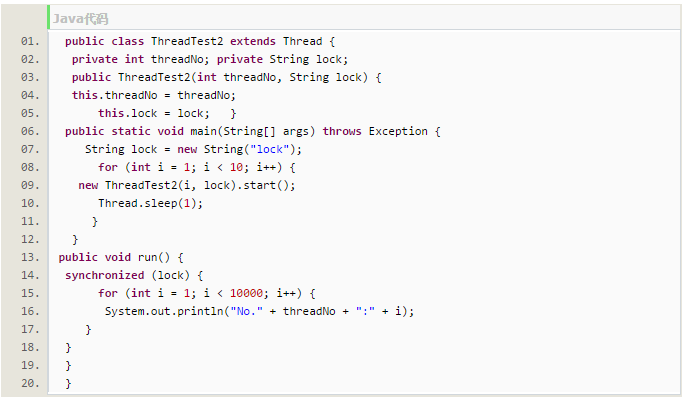


分析：

（1）、这个程序其实就是让10个线程在控制台上数数，从1数到9999。理想情况下，我们希望看到一个线程数完，然后才是另一个线程开始数数。

（2）、但是这个程序的执行过程告诉我们，这些线程还是乱糟糟的在那里抢着报数，丝毫没有任何规矩可言。  
     （3）、但是run方法还是加了一个synchronized关键字的，按道理说，这些线程应该可以一个接一个的执行这个run方法才对阿。  
     （4）、上面我们提到：对于一个成员方法加synchronized关键字，这实际上是以这个成员方法所在的对象本身作为对象锁。在本例中，就是以ThreadTest类的一个具体对象：该线程自身作为对象锁的。一共十个线程，每个线程持有自己线程对象的那个对象锁。这必然不能产生同步的效果。换句话说，**如果要对这些线程进行同步，那么这些线程所持有的对象锁应当是共享且唯一的!**

(2)、使用lock + synchronized实现多线程同步



分析：

  （1）、该程序通过在main方法启动10个线程之前，创建了一个String类型的对象。并通过ThreadTest2的构造函数，将这个对象赋值给每一个ThreadTest2线程对象中的私有变量lock。

（2）、根据Java方法的传值特点，这些线程的lock变量实际上指向的是堆内存中的同一个区域，即存放main函数中的lock变量的区域。  
       （3）、程序将原来run方法前的synchronized关键字去掉，换用了run方法中的一个synchronized块来实现。这个同步块的对象锁，就是 main方 法中创建的那个String对象。换句话说，他们指向的是同一个String类型的对象，对象锁是共享且唯一的！

（3）、使用静态方法实现多线程同步



分析：

（1）、这段代码没有使用main方法中创建的String对象作为这10个线程的线程锁。而是通过在run方法中调用本线程中一个静态的同步方法abc而实现了线程的同步。  
 （2）、synchronized静态方法是用什么来做对象锁的呢？ 对于同步静态方法，对象锁就是该静态放发所在的类的Class实例。由于在JVM中，所有被加载的类都有唯一的类对象，具体到本例就是唯一的ThreadTest3.class对象。不管我们创建了该类的多少实例，但是它的类实例仍然是一个！

（4）、总结

（1）、对于同步的方法或者代码块来说，必须获得对象锁才能够进入同步方法或者代码块进行操作；

（2）、如果采用method级别的同步，则对象锁即为method所在的对象；如果是静态方法，对象锁即指method所在的Class对象(唯一)；

（3）、对于代码块，对象锁即指synchronized(lock)中的lock；

（4）、第一种情况：对象锁即为每一个线程对象，因此有多个，所以同步失效；

第二种情况：共用同一个对象锁lock，因此同步生效；

第三种情况：因为是static，所以对象锁为ThreadTest3的class对象（唯一），因此同步生效。

（5）、同步有两种方式，同步块和同步方法（为什么没有wait和notify？这个我会在补充章节中做出阐述）  
 （5.1）、如果是同步代码块，则对象锁需要编程人员自己指定，一般有些代码为synchronized(this)只有在单态模式才生效；  
 （5.2）、如果是同步方法，**则分静态和非静态两种** 。  
 静态方法则一定会同步，非静态方法需在单例模式才生效，推荐用静态方法(不用担心是否单例)。  
 所以说，在Java多线程编程中，最常见的synchronized关键字实际上是依靠对象锁的机制来实现线程同步的。我们似乎可以听到synchronized在向我们说：“给我**一把** 锁，我能创造一个规矩”。

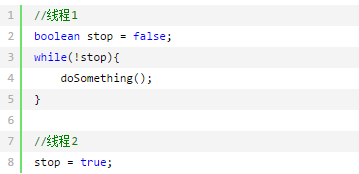
Volatile关键字： <http://www.cnblogs.com/dolphin0520/p/3920373.html>

1、一旦一个共享变量（类的成员变量、类的静态成员变量）被volatile修饰之后，那么就具备了两层语义：

1）、保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这新值对其他线程来说是立即可见的。

2）、禁止进行指令重排序。

举例：



这段代码为何有可能导致无法中断线程：

前面提到每个线程在运行过程中都有自己的工作内存，那么线程1在运行的时候，会将stop变量的值拷贝一份放在自己的工作内存当中。那么当线程2更改了stop变量的值之后，但是还没来得及写入主存当中，线程2转去做其他事情了，那么线程1由于不知道线程2对stop变量的更改，因此还会一直循环下去。

但是用volatile修饰之后就变得不一样了：

1、保证了操作的可见性：

第一：使用volatile关键字会强制将修改的值立即写入主存；

第二：使用volatile关键字的话，当线程2进行修改时，会导致线程1的工作内存中缓存变量stop的缓存行无效

（反映到硬件层的话，就是CPU的L1或者L2缓存中对应的缓存行无效）；

第三：由于线程1的工作内存中缓存变量stop的缓存行无效，所以线程1再次读取变量stop的值时会去主存读取。

总结：

那么在线程2修改stop值时（当然这里包括2个操作，修改线程2工作内存中的值，然后将修改后的值写入内存），会使得线程1的工作内存中缓存变量stop的缓存行无效，然后线程1读取时，发现自己的缓存行无效，它会等待缓存行对应的主存地址被更新之后，然后去对应的主存读取最新的值。那么线程1读取到的就是最新的正确的值。

2、volatile也无法保证对变量的任何操作都是原子性的： 使用synchronized 或则lock都可以修正。

3、volatile 能在一定程度上保证有序性。

volatile关键字禁止指令重排序有两层意思：

1）当程序执行到volatile变量的读操作或者写操作时，在其前面的操作的更改肯定全部已经进行，且结果已经对后面的操作可见；在其后面的操作肯定还没有进行；

2）在进行指令优化时，不能将在对volatile变量访问的语句放在其后面执行，也不能把volatile变量后面的语句放到其前面执行。

4、volatile的应用场景：

　synchronized关键字是防止多个线程同时执行一段代码，那么就会很影响程序执行效率；

volatile关键字在某些情况下性能要优于synchronized，但是要注意volatile关键字是无法替代synchronized关键字的，因为volatile关键字无法保证操作的原子性。

通常来说，使用volatile必须具备以下2个条件：

　　 1）对变量的写操作不依赖于当前值 i++ 就不行

　　 2）该变量没有包含在具有其他变量的不变式中 i= j + 1 就不行

　　实际上，这些条件表明，可以被写入 volatile 变量的这些有效值独立于任何程序的状态，包括变量的当前状态。

　　事实上，我的理解就是上面的2个条件需要保证操作是原子性操作，才能保证使用volatile关键字的程序在并发时能够正确执行。

下面列举几个Java中使用volatile的几个场景。

**1）、状态标记量；2）double check**

**四、Java中的原始数据类型都有哪些，它们的大小及对应的封装类是什么？**

参考网址：http://www.cnblogs.com/BillLei/p/5568917.html

**（1）boolean --** 封装类是Boolean

   boolean数据类型非true即false。这个数据类型表示1 bit的信息，但是它的大小并没有精确定义；这样我们可以得出boolean类型单独使用是4个字节，在数组中又是1个字节。

那虚拟机为什么要用int来代替boolean呢？为什么不用byte或short，这样不是更节省内存空间吗？实际上，使用int的原因是，对于当下32位的CPU来说，一次进行32位的数据交换更加高效。

综上，我们可以知道：官方文档对boolean类型没有给出精确的定义，《Java虚拟机规范》给出了“单独时使用4个字节，boolean数组时1个字节”的定义，具体还要看虚拟机实现是否按照规范来，所以1个字节、4个字节都是有可能的。这其实是一种时空权衡。

（2）byte——1 byte——Byte

（3）char——2 bytes——Character

（4）short——2 bytes——Short

（5）int——4 bytes——Integer

（6）float——4 bytes——Float

（7）long——8 bytes——Long

（8）double——8 bytes——Double

进程和线程

1.进程和线程有什么区别？总结起来，就是一下的几个区别：

　　a.进程是资源分配的基本单位，线程是cpu调度，或者说是程序执行的最小单位。因此，实现并发功能的单位是线程。

　 b.进程有独立的地址空间（比如在linux下面启动一个新的进程，系统必须分配给它独立的地址空间，建立众多的数据表来维护它的代码段、堆栈段和数据段，这是一种非常昂贵的多任务工作方式）。而运行一个进程中的线程，它们之间共享大部分数据，使用相同的地址空间，因此启动一个线程，切换一个线程远比进程操作要快，花费也要小得多。当然，线程是拥有自己的局部变量和堆栈（注意不是堆）的（比如在windows中用[\_beginthreadex](http://blog.csdn.net/morewindows/article/details/7421759)创建一个新进程就会在调用CreateThread的同时申请一个专属于线程的数据块（\_tiddata)）。

　　c.线程之间的通信比较方便。同一个进程下的线程共享数据（比如全局变量，静态变量），通过这些数据来通信不仅快捷而且方便，当然如何处理好这些访问的同步与互斥正是编写多线程程序的难点。而进程之间的通信只能通过[进程通信](http://baike.baidu.com/view/549640.htm)的方式进行。

　　d. 由b可轻易地得到结论：多进程比多线程程序要健壮。一个线程死掉整个进程就死掉了，但是在[保护模式](http://blog.csdn.net/wukaiyu/article/details/1766804)下，一个进程死掉对另一个进程没有直接影响。

e.线程的执行与进程是有区别的。每个独立的进程有有自己的一个程序入口，顺序执行序列和程序的出口，但是线程不能独立执行，必须依附与程序之中，由应用程序提供多个线程的并发控制。

2. 什么是线程安全？ 如果多线程的程序运行结果是可预期的，而且与单线程的程序运行结果一样，那么说明是“线程安全”的。

[5、进程间的通信方式](http://www.cnblogs.com/mydomain/archive/2010/09/23/1833369.html)

管道：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。  
有名管道：有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

信号量：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问

资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

消息队列： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及

缓冲区大小受限等缺点。

信号 ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

共享内存：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方

式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。

套接字： 套解口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

数据库优化

1、**选择最有效率的表名顺序:** 在FROM子句中包含多个表的情况下,因为执行顺序为从后往前, 所以数据量较少的表尽量放后。

2、**WHERE子句中的连接顺序：因为**自下而上的顺序解析WHERE子句,所以表之间的连接必须写在其他WHERE条件之前, 那些可以过滤掉最大数量记录的条件必须写在WHERE子句的末尾.

3、**SELECT子句中避免使用 ‘ \* ‘：**在解析过程中会将'\*' 依次转换成列名, 这个工作是通过查询数据字典完成的, 这意味着将耗费更多的时间。

4、**减少访问数据库的次数： 在工作中的经验就是for循环中尽量不能放sql语句，在for之前一次性查出，在循环组装数据。**

5、**用Where子句替换HAVING子句。**

6、**减少对表的查询：**在含有子查询的SQL语句中,要特别注意减少对表的查询.例子：

7、**通过内部函数提高SQL效率：**复杂的SQL往往牺牲了执行效率. 能够掌握上面的运用函数解决问题。

8、**使用表的别名(Alias)**

9、**用EXISTS替代IN、用NOT EXISTS替代NOT IN。**

在子查询中,NOT IN子句将执行一个内部的排序和合并. 无论在哪种情况下,NOT IN都是最低效的：因为它对子查询中的表执行了一个全表遍历。

10、**用索引提高效率：**

11、**用EXISTS替换DISTINCT：**  
当提交一个包含一对多表信息(比如部门表和雇员表)的查询时,避免在SELECT子句中使用DISTINCT. 一般可以考虑用EXIST替换, EXISTS 使查询更为迅速,因为RDBMS核心模块将在子查询的条件一旦满足后,立刻返回结果.

12、**在java代码中尽量少用连接符“＋”连接字符串！**

13、**避免在索引列上使用NOT ：通常**NOT会产生在和在索引列上使用函数相同的影响. 当ORACLE”遇到”NOT,他就会停止使用索引转而执行全表扫描.

14、**避免在索引列上使用计算．**WHERE子句中，如果索引列是函数的一部分．优化器将不使用索引而使用全表扫描

15、**用>=替代>**

16、用IN来替换OR

<http://www.jb51.net/article/19024.htm>

数据库索引

0、索引的实现通常使用B树及其变种B+树。

1、建立索引的优点：

1. 创建唯一性索引，保证数据库表中每一行数据的唯一性;

2. 大大加快数据的检索速度;

3.加速表和表之间的连接;

……

2、建立索引的目的是加快对表中记录的查找或排序。

3、缺点：

一、创建索引和维护索引要耗费时间，这种时间随着数据量的增加而增加；

二、是增加了数据库的存储空间；

三、是在插入和修改数据时要花费较多的时间(因为索引也要随之变动)。

4、索引类型：

4.1、唯一索引：唯一索引是不允许其中任何两行具有相同索引值的索引。

当现有数据中存在重复的键值时，大多数数据库不允许将新创建的唯一索引与表一起保存。数据库还可能防止添加将在表中创建重复键值的新数据。例如，如果在employee表中职员的姓(lname)上创建了唯一索引，则任何两个员工都不能同姓。

4.2、主键索引

数据库表经常有一列或列组合，其值唯一标识表中的每一行。该列称为表的主键。

在数据库关系图中为表定义主键将自动创建主键索引，主键索引是唯一索引的特定类型。该索引要求主键中的每个值都唯一。当在查询中使用主键索引时，它还允许对数据的快速访问。

4.3、聚集索引

在聚集索引中，表中行的物理顺序与键值的逻辑（索引）顺序相同。一个表只能包含一个聚集索引。

如果某索引不是聚集索引，则表中行的物理顺序与键值的逻辑顺序不匹配。与非聚集索引相比，聚集索引通常提供更快的数据访问速度。

 创建索引:  
　  单一索引:Create Index <Index-Name> On<Table\_Name>(Column\_Name);

　  复合索引: Create Index i\_deptno\_job onemp(deptno,job); —>在emp表的deptno、job列建立索引。

      DBA经常用 REBUILD 来重建索引可以减少硬盘碎片和提高应用系统的性能。

      ALTER INDEX emp\_ix REBUILD REVERSE;   //修改索引

      drop index pk\_dept;     //删除索引

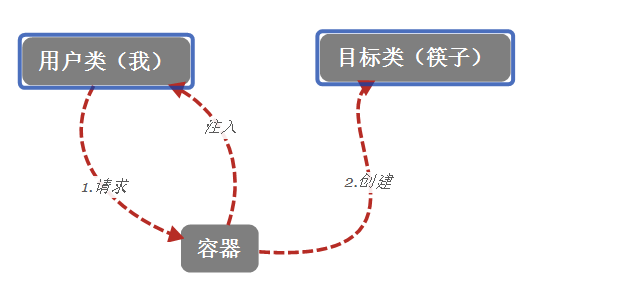
TCP/IP知识点

JAVA中SSM框架的准备

一、IOC：inverse of Control：控制反转。 <http://www.cnblogs.com/xxzhuang/p/5948902.html>

程序中的之间的关系，不用代码控制，而完全是由容器来控制。在运行阶段，容器会根据配置信息直接把他们的关系注入到组件中。

依赖注入和控制反转其实是一个概念。只不过强调的不同而已，依赖注入强调关系的注入是由容器在运行时完成，而控制反转强调关系是由容器控制。其实本质是一样的。



在控制反转中获取资源的过程叫做依赖注入，那么这里代码实现也是专注于依赖注入。

依赖注入有3种方式，分别为构造注入，设置注入，接口注入（spring不支持）。

事务(Transaction)是访问并可能更新数据库中各种数据项的一个程序执行单元(unit)。