线程安全（同步）线程不安全（非同步）

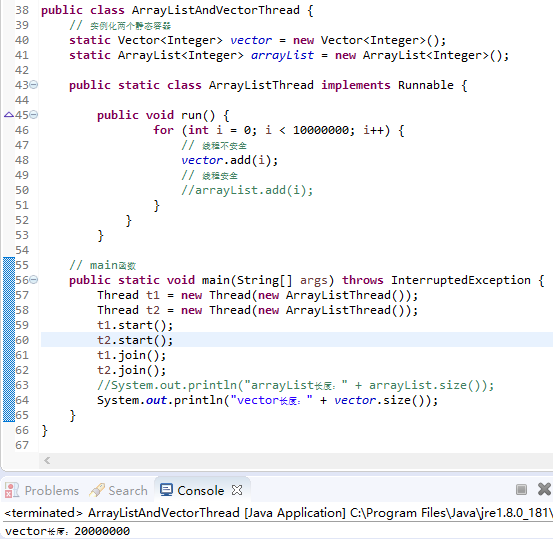
1 Java中的集合

Java中的集合分为同步的集合(线程安全)和线程不安全的集合 例如 :

ArrayList和Vector的区别：

一.同步性:Vector是线程安全的，也就是说是同步的，而ArrayList是线程序不安全的，不是同步的

二.数据增长:当需要增长时,Vector默认增长为原来一倍，而ArrayList默认增长为原来的1.5倍



HashMap和Hashtable的区别

一.历史原因:Hashtable是基于陈旧的Dictionary类的，HashMap是Java 1.2引进的Map接口的一个实现

二.同步性:Hashtable是线程安全的，也就是说是同步的，效率低一些；而HashMap是线程序不安全的，不是同步的 效率高

三.值： HashMap允许1个key为null 多个value为null  Hashtable key value都不能为null

在Java的集合容器框架中，主要有四大类别：List、Set、Queue、Map。

List、Set、Queue接口分别继承了Collection接口，Map本身是一个接口。

注意Collection和Map是一个顶层接口，而List、Set、Queue则继承了Collection接口，分别代表数组、集合和队列这三大类容器。

像ArrayList、LinkedList都是实现了List接口，HashSet实现了Set接口，而Deque（双向队列，允许在队首、队尾进行入队和出队操作）继承了Queue接口，PriorityQueue实现了Queue接口。

另外LinkedList（实际上是双向链表）实现了了Deque接口。

像ArrayList、LinkedList、HashMap这些容器都是非线程安全的。

如果有多个线程并发地访问这些容器时，就会出现问题。

因此，在编写程序时，我们必须手动地在任何访问到这些容器的地方进行同步处理（但是有的就算是进行了同步也不一定会安全）。

ArrayList和Vector的详解  
我需要思考的是Vector 是线程安全的 (在任何情况下都是)。。。吗？  
  
觉得是原因可能是因为 Vector 的所有方法加上了 synchronized 关键字，从而保证访问 vector 的任何方法都必须获得对象的钥匙 intrinsic lock (或叫 monitor lock)，也即，在vector内部，其所有方法不会被多线程所访问。  
但是，以下代码呢：  
if (!vector.contains(element))   
    vector.add(element);   
    ...  
}  
  
这是经典的 put-if-absent 情况，尽管 contains, add 方法都正确地同步了，但作为 vector 之外的使用环境，仍然存在race condition: 因为虽然条件判断 if (!vector.contains(element))与方法调用 vector.add(element);  都是自动性的操作 (atomic)，但在 if 条件判断为真后，那个用来访问vector.contains 方法的锁已经释放，在即将的 vector.add 方法调用 之间有间隙，在多线程环境中，完全有可能被其他线程获得 vector的 lock 并改变其状态, 此时当前线程的vector.add(element);  正在等待（只不过我们不知道而已）。只有当其他线程释放了 vector 的 lock 后，vector.add(element); 继续，但此时它已经基于一个错误的假设了。  
  
单个的方法 synchronized 了并不代表组合（compound）的方法调用具有原子性，使 compound actions  成为线程安全的可能解决办法之一还是离不开intrinsic lock本身锁的 (这个锁应该是 vector 的，但由 client 维护)：  
  
// Vector v = ...  
    public  boolean putIfAbsent(E x) {  
synchronized(v) {   
            boolean absent = !contains(x);   
            if (absent) {   
                add(x);  
}   
}  
        return absent;   
    }  
所以，  
Vector 和 ArrayList 实现了同一接口 List, 但所有的 Vector 的方法都具有 synchronized 关键修饰。但对于**复合操作**，Vector 仍然需要进行同步处理。   
  
这样做的后果，Vector 应该尽早地被废除，因为这样做本身没有解决多线程问题，反而，在引入了概念的混乱的同时，导致性能问题，因为 synchronized 的开销是巨大的：阻止编译器乱序，hint for 处理器寄存一/二级缓存。。。