List

一、ArrayList,Vector, LinkedList的存储性能和特性.

ArrayList和Vector都是使用数组方式存储数据，此数组元素数大于实际存储的数据以便增加和插入元素，它们都允许直接按序号索引元素，但是插入元素要涉及数组元素移动等内存操作，所以索引数据快而插入数据慢

Vector由于使用了synchronized方法（线程安全），通常性能上较ArrayList差，

LinkedList使用双向链表实现存储，按序号索引数据需要进行前向或后向遍历，但是插入数据时只需要记录本项的前后项即可，所以插入速度较快。

如果查找一个指定位置的数据，vector和arraylist使用的时间是相同的，都是0(1),这个时候使用vector和arraylist都可以。

而如果移动一个指定位置的数据花费的时间为0(n-i)n为总长度，这个时候就应该考虑到使用linklist,因为它移动一个指定位置的数据所花费的时间为0(1),而查询一个指定位置的数据时花费的时间为0(i)。

二、Vector与ArrayList的区别

1)vector是线程同步的，所以它也是线程安全的，而arraylist是线程异步的，是不安全的。如果不考虑到线程的安全因素，一般用arraylist效率比较高。

2)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%+1个

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。

3)Vector提供indexOf(obj, start)接口，ArrayList没有。

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。

```
if (!vector.contains(element))
    vector.add(element);
    ...
}
```

虽然条件判断 if (!vector.contains(element))与方法调用 vector.add(element); 都是原子性的操作 (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