一般大家都知道ArrayList和LinkedList的大致区别：   
     1.ArrayList是实现了基于动态数组的**[数据结构](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**，LinkedList基于链表的数据结构。   
     2.对于随机访问get和set，ArrayList觉得优于LinkedList，因为LinkedList要移动指针。   
     3.对于新增和删除操作add和remove，LinedList比较占优势，因为ArrayList要移动数据。   
  
ArrayList和LinkedList是两个集合类，用于存储一系列的对象引用(references)。例如我们可以用ArrayList来存储一系列的String或者Integer。那么ArrayList和LinkedList在性能上有什么差别呢？什么时候应该用ArrayList什么时候又该用LinkedList呢？

一．时间复杂度   
首先一点关键的是，ArrayList的内部实现是基于基础的对象数组的，因此，它使用get方法访问列表中的任意一个元素时(random access)，它的速度要比LinkedList快。LinkedList中的get方法是按照顺序从列表的一端开始检查，直到另外一端。对LinkedList而言，访问列表中的某个指定元素没有更快的方法了。   
假设我们有一个很大的列表，它里面的元素已经排好序了，这个列表可能是ArrayList类型的也可能是LinkedList类型的，现在我们对这个列表来进行二分查找(binary search)，比较列表是ArrayList和LinkedList时的查询速度，看下面的程序：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | package com.mangocity.test;  import java.util.LinkedList;  import java.util.List;  import java.util.Random;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.Collections;  public class TestList ...{       public static final int N=50000;         public static List values;         static...{           Integer vals[]=new Integer[N];             Random r=new Random();             for(int i=0,currval=0;i<N;i++)...{               vals=new Integer(currval);               currval+=r.nextInt(100)+1;           }             values=Arrays.asList(vals);       }         static long timeList(List lst)...{           long start=System.currentTimeMillis();           for(int i=0;i<N;i++)...{               int index=Collections.binarySearch(lst, values.get(i));               if(index!=i)                   System.out.println("\*\*\*错误\*\*\*");           }           return System.currentTimeMillis()-start;       }       public static void main(String args[])...{           System.out.println("ArrayList消耗时间："+timeList(new ArrayList(values)));           System.out.println("LinkedList消耗时间："+timeList(new LinkedList(values)));       }  } |

　　我得到的输出是：ArrayList消耗时间：15   
                            LinkedList消耗时间：2596   
这个结果不是固定的，但是基本上ArrayList的时间要明显小于LinkedList的时间。因此在这种情况下不宜用LinkedList。二分查找法使用的随机访问(random access)策略，而LinkedList是不支持快速的随机访问的。对一个LinkedList做随机访问所消耗的时间与这个list的大小是成比例的。而相应的，在ArrayList中进行随机访问所消耗的时间是固定的。   
这是否表明ArrayList总是比LinkedList性能要好呢？这并不一定，在某些情况下LinkedList的表现要优于ArrayList，有些**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "_blank)**在LinkedList中实现时效率更高。比方说，利用Collections.reverse方法对列表进行反转时，其性能就要好些。   
看这样一个例子，加入我们有一个列表，要对其进行大量的插入和删除操作，在这种情况下LinkedList就是一个较好的选择。请看如下一个极端的例子，我们重复的在一个列表的开端插入一个元素：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | package com.mangocity.test;    import java.util.\*;  public class ListDemo {       static final int N=50000;       static long timeList(List list){       long start=System.currentTimeMillis();       Object o = new Object();       for(int i=0;i<N;i++)           list.add(0, o);       return System.currentTimeMillis()-start;       }       public static void main(String[] args) {           System.out.println("ArrayList耗时："+timeList(new ArrayList()));           System.out.println("LinkedList耗时："+timeList(new LinkedList()));       }  } |

这时我的输出结果是：ArrayList耗时：2463

                            LinkedList耗时：15   
这和前面一个例子的结果截然相反，当一个元素被加到ArrayList的最开端时，所有已经存在的元素都会后移，这就意味着数据移动和复制上的开销。相反的，将一个元素加到LinkedList的最开端只是简单的未这个元素分配一个记录，然后调整两个连接。在LinkedList的开端增加一个元素的开销是固定的，而在ArrayList的开端增加一个元素的开销是与ArrayList的大小成比例的。

二．空间复杂度   
在LinkedList中有一个私有的内部类，定义如下：

|  |  |
| --- | --- |
| 1  2  3  4  5 | private static class Entry {           Object element;           Entry next;           Entry previous;       } |

每个Entry对象reference列表中的一个元素，同时还有在LinkedList中它的上一个元素和下一个元素。一个有1000个元素的LinkedList对象将有1000个链接在一起的Entry对象，每个对象都对应于列表中的一个元素。这样的话，在一个LinkedList结构中将有一个很大的空间开销，因为它要存储这1000个Entity对象的相关信息。   
ArrayList使用一个内置的数组来存储元素，这个数组的起始容量是10.当数组需要增长时，新的容量按如下公式获得：新容量=(旧容量\*3)/2+1，也就是说每一次容量大概会增长50%。这就意味着，如果你有一个包含大量元素的ArrayList对象，那么最终将有很大的空间会被浪费掉，这个浪费是由ArrayList的工作方式本身造成的。如果没有足够的空间来存放新的元素，数组将不得不被重新进行分配以便能够增加新的元素。对数组进行重新分配，将会导致性能急剧下降。如果我们知道一个ArrayList将会有多少个元素，我们可以通过构造方法来指定容量。我们还可以通过trimToSize方法在ArrayList分配完毕之后去掉浪费掉的空间。

三．总结   
ArrayList和LinkedList在性能上各有优缺点，都有各自所适用的地方，总的说来可以描述如下：   
1．对ArrayList和LinkedList而言，在列表末尾增加一个元素所花的开销都是固定的。对ArrayList而言，主要是在内部数组中增加一项，指向所添加的元素，偶尔可能会导致对数组重新进行分配；而对LinkedList而言，这个开销是统一的，分配一个内部Entry对象。

2．在ArrayList的中间插入或删除一个元素意味着这个列表中剩余的元素都会被移动；而在LinkedList的中间插入或删除一个元素的开销是固定的。

3．LinkedList不支持高效的随机元素访问。

4．ArrayList的空间浪费主要体现在在list列表的结尾预留一定的容量空间，而LinkedList的空间花费则体现在它的每一个元素都需要消耗相当的空间

可以这样说：当操作是在一列数据的后面添加数据而不是在前面或中间,并且需要随机地访问其中的元素时,使用ArrayList会提供比较好的性能；当你的操作是在一列数据的前面或中间添加或删除数据,并且按照顺序访问其中的元素时,就应该使用LinkedList了。

tips：

ArrayList 是线性表（数组）  
get() 直接读取第几个下标，复杂度 O(1)  
add(E) 添加元素，直接在后面添加，复杂度O（1）  
add(index, E) 添加元素，在第几个元素后面插入，后面的元素需要向后移动，复杂度O（n）  
remove（）删除元素，后面的元素需要逐个移动，复杂度O（n）  
  
LinkedList 是链表的操作  
get() 获取第几个元素，依次遍历，复杂度O(n)  
add(E) 添加到末尾，复杂度O(1)  
add(index, E) 添加第几个元素后，需要先查找到第几个元素，直接指针指向操作，复杂度O(n)  
remove（）删除元素，直接指针指向操作，复杂度O(1)