# Java中ArrayList 、LinkList的区别分析

1.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。   
     2.对于随机访问get和set，ArrayList优于LinkedList，因为ArrayList可以随机定位，而LinkedList要移动指针一步一步的移动到节点处。（参考数组与链表来思考）  
     3.对于新增和删除操作add和remove，LinedList比较占优势，只需要对指针进行修改即可，而ArrayList要移动数据来填补被删除的对象的空间。

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：

复制代码代码如下:

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(randomaccess)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，假如我们有一个列表，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：

复制代码代码如下:

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为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：

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：

性能总结：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **-** | **add()操作** | **delete()操作** | **insert操作** | **index取值操作** | **iterator取值操作** |
| ArrayList/Vector/Stack | 好 | 差 | 差 | 极优 | 极优 |
| LinkedList | 好 | 好 | 好 | 差 | 极优 |

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。

java中ArrayList 、List区别

List集合  
    List继承自Collection接口。List是一种有序集合，List中的元素可以根据索引（顺序号：元素在集合中处于的位置信息）进行取得/删除/插入操作。

    跟Set集合不同的是，List允许有重复元素。对于满足e1.equals(e2)条件的e1与e2对象元素，可以同时存在于List集合中。当然，也有List的实现类不允许重复元素的存在。  
   同时，List还提供一个listIterator()方法，返回一个ListIterator接口对象，和Iterator接口相比，ListIterator添加元素的添加，删除，和设定等方法，还能向前或向后遍历。

List跟Collection的关系：  
java.util.Collection [I]  
+--java.util.List [I]  
   +--java.util.ArrayList [C]  
   +--java.util.LinkedList [C]  
   +--java.util.Vector [C]  
      +--java.util.Stack [C]

List接口的实现类主要有ArrayList，LinkedList，Vector，Stack等。

父子关系.  
   List是一个接口,ArrayList继承与这个接口并实现了它.  
   用的时候一般都用ArrayList.没用过List. 可以这么用:List list = new ArrayList();

Collection接口  
    Collection是最基本的集合接口，一个Collection代表一组Object，即Collection的元素（Elements）。一些Collection允许相同的元素而另一些不行。一些能排序而另一些不行。Java SDK不提供直接继承自Collection的类，JavaSDK提供的类都是继承自Collection的“子接口”如List和Set。  
    所有实现Collection接口的类都必须提供两个标准的构造函数：无参数的构造函数用于创建一个空的Collection，有一个Collection参数的构造函数用于创建一个新的Collection，这个新的Collection与传入的Collection有相同的元素。后一个构造函数允许用户复制一个Collection。

     如何遍历Collection中的每一个元素？不论Collection的实际类型如何，它都支持一个iterator()的方法，该方法返回一个迭代子，使用该迭代子即可逐一访问Collection中每一个元素。典型的用法如下：  
    Iterator it = collection.iterator(); // 获得一个迭代子  
    while(it.hasNext()) {  
                             Object obj = it.next(); // 得到下一个元素  
       }  
由Collection接口派生的两个接口是List和Set。

List接口：  
    List是有序的Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在List中的位置，类似于数组下标）来访问List中的元素，这类似于Java的数组。  
和下面要提到的Set不同，List允许有相同的元素。  
除了具有Collection接口必备的iterator()方法外，List还提供一个listIterator()方法，返回一个ListIterator接口，和标准的Iterator接口相比，ListIterator多了一些add()之类的方法，允许添加，删除，设定元素，还能向前或向后遍历。

    实现List接口的常用类有LinkedList，ArrayList，Vector和Stack。

LinkedList类  
     LinkedList实现了List接口，允许null元素。此外LinkedList提供额外的get，remove，insert方法在LinkedList的首部或尾部。这些操作使LinkedList可被用作堆栈（stack），队列（queue）或双向队列（deque）。  
    注意LinkedList没有同步方法。如果多个线程同时访问一个List，则必须自己实现访问同步。一种解决方法是在创建List时构造一个同步的List：  
List list = Collections.synchronizedList(new LinkedList(...));

ArrayList类  
ArrayList实现了可变大小的数组。它允许所有元素，包括null。ArrayList没有同步。  
size，isEmpty，get，set方法运行时间为常数。但是add方法开销为分摊的常数，添加n个元素需要O(n)的时间。其他的方法运行时间为线性。  
每个ArrayList实例都有一个容量（Capacity），即用于存储元素的数组的大小。这个容量可随着不断添加新元素而自动增加，但是增长算法并没有定义。当需要插入大量元素时，在插入前可以调用ensureCapacity方法来增加ArrayList的容量以提高插入效率。  
和LinkedList一样，ArrayList也是非同步的（unsynchronized）。

总结  
　　如果涉及到堆栈，队列等操作，应该考虑用List，对于需要快速插入，删除元素，应该使用LinkedList，如果需要快速随机访问元素，应该使用ArrayList。  
      尽量返回接口而非实际的类型，如返回List而非ArrayList，这样如果以后需要将ArrayList换成LinkedList时，客户端代码不用改变。这就是针对抽象编程。