[ArrayList和LinkedList的用法区别：](http://blog.csdn.net/h11h03/article/details/3157968)

标签： [string](http://www.csdn.net/tag/string)[insert](http://www.csdn.net/tag/insert)[数据结构](http://www.csdn.net/tag/%e6%95%b0%e6%8d%ae%e7%bb%93%e6%9e%84)[session](http://www.csdn.net/tag/session)[语言](http://www.csdn.net/tag/%e8%af%ad%e8%a8%80)[vb](http://www.csdn.net/tag/vb)

2008-10-27 13:46 10646人阅读 [评论](http://blog.csdn.net/h11h03/article/details/3157968#comments)(1) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/h11h03/article/details/3157968#report)

版权声明：本文为博主原创文章，未经博主允许不得转载。

 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。  
    这一点要看实际情况的。若只对单条数据插入或删除，ArrayList的速度反而优于LinkedList。但若是批量随机的插入删除数据，LinkedList的速度大大优于ArrayList. 因为ArrayList每插入一条数据，要移动插入点及之后的所有数据。  这一点我做了实验。在分别有200000条“记录”的ArrayList和LinkedList的首位插入20000条数据，LinkedList耗时约是ArrayList的20分之1。

http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockStart.giffor(int m=0;m<20000;m++){  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        linkedlist.add(m,null);      //当在200000条数据之前插入20000条数据时，LinkedList只用了1125多ms.这就是LinkedList的优势所在  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockEnd.gif        }  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        long time4 = new Dte().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        System.out.print("batch linkedlist add:");  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        System.out.println(time4 - time3);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockStart.gif        for(int n=0;n<20000;n++){  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        arraylist.add(n, null);  //当在200000条数据之前插入20000条数据时，ArrayList用了18375多ms.时间花费是arraylist的近20倍(视测试时机器性能)  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockEnd.gif        }  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        long time5 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        System.out.print("batch arraylist add:");  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        System.out.println(time5 - time4);

4.查找操作indexOf,lastIndexOf,contains等，两者差不多。  
5.随机查找指定节点的操作get，ArrayList速度要快于LinkedList.  
这里只是理论上分析，事实上也不一定，ArrayList在末尾插入和删除数据的话，速度反而比LinkedList要快。我做过一个插入和删除200000条数据的试验。

http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        long time1 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        String s1 = (String) linkedlist.get(100000);//  总记录200000，linkedlist加载第100000条数据耗时15~32ms不等  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        long time2 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        System.out.println(time2 - time1);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        String s2 = (String) arraylist.get(100000);//  总记录200000，linkedlist加载第100000条数据耗时0ms  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        long time3 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif        System.out.println(time3 - time2);

http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockStart.gif    /\*分别insert200000条数据到linkedlist和arraylist  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif    \*由于是在末尾插入数据，arraylist的速度比linkedlist的速度反而要快   
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockEnd.gif    \*/  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockStart.gif    public static void insertList(LinkedList linklist, ArrayList arraylist) {  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        long time1 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        System.out.println(time1);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedSubBlockStart.gif        for (int i = 0; i < 200000; i++) {  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif            linklist.add(i, "linklist" + i);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        long time2 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        System.out.println(time2 - time1);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedSubBlockStart.gif        for (int j = 0; j < 200000; j++) {  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif            arraylist.add(j, "arraylist" + j);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        long time3 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        System.out.println(time3 - time2);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockEnd.gif    }  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockStart.gif/\*delete linkedlist和arraylist中的200000条数据  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif    \*由于是在末尾删除数据，arraylist的速度比linkedlist的速度反而要快   
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockEnd.gif    \*/  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockStart.gif    public static void deleteList(LinkedList linklist, ArrayList arraylist) {  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        long time1 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        System.out.println(time1);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedSubBlockStart.gif        for (int i = 199999; i >= 0; i--) {  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif            linklist.remove(i);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        long time2 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        System.out.println(time2 - time1);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedSubBlockStart.gif        for (int j = 199999; j >= 0; j--) {  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif            arraylist.remove(j);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        long time3 = new Date().getTime();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        System.out.println(time3 - time2);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockEnd.gif    }  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/None.gif  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/ExpandedBlockStart.gif    public static void main(String args[]) {  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        LinkedList linkedlist = new LinkedList();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        ArrayList arraylist = new ArrayList();  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif        insertList(linkedlist, arraylist);  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif  
http://images.csdn.net/syntaxhighlighting/OutliningIndicators/InBlock.gif                                        //以下代码省略

插入：  
LinkedList 578ms  
ArrayList 437ms  
删除：  
LinkedList 31ms  
ArrayList 16ms

**比较一下ArrayList和LinkedList：  
1.ArrayList是基于数组，LinkedList基于链表实现。  
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。  
4.查找操作indexOf,lastIndexOf,contains等，两者差不多。  
这里只是理论上分析，事实上也不一定，比如ArrayList在末尾插入和删除数据就不设计到数据移动，不过还是  
有这么个建议：随机访问比较多的话一定要用ArrayList而不是LinkedList，如果需要频繁的插入和删除应该  
考虑用LinkedList来提高性能。**

**如果z 引用一个当前内容是“start”的字符串缓冲区对象，则此方法调用 z.append("le") 会使字符串缓冲区包含“startle”，而 z.insert(4, "le") 将更改字符串缓冲区，使之包含“starlet”。**

**每个字符串缓冲区都有一定的容量。只要字符串缓冲区所包含的字符序列的长度没有超出此容量，就无需分配新的内部缓冲区数组。如果内部缓冲区溢出，则此容量自动增大。从 JDK 5 开始，为该类补充了一个单个线程使用的等价类，即 StringBuilder。与该类相比，通常应该优先使用 StringBuilder 类，因为它支持所有相同的操作，但由于它不执行同步，所以速度更快。 例如：如果 sb 引用 StringBuilder 的一个实例，则 sb.append(x) 和 sb.insert(sb.length(), x) 具有相同的效果。java.lang.StringBuilder一个可变的字符序列。此类提供一个与 StringBuffer 兼容的 API，但不保证同步。该类被设计用作 StringBuffer 的一个简易替换，用在字符串缓冲区被单个线程使用的时候（这种情况很普遍）。如果可能，建议优先采用该类，因为在大多数实现中，它比 StringBuffer 要快。在 StringBuilder 上的主要操作是 append 和 insert 方法，可重载这些方法，以接受任意类型的数据。每个方法都能有效地将给定的数据转换成字符串，然后将该字符串的字符追加或插入到字符串生成器中。 append 方法始终将这些字符添加到生成器的末端；而 insert 方法则在指定的点添加字符。   
  
String / StringBuffer /StringBuilder，在字符串连接操作上性能依次加强，我曾经做过一个实验，连续进行10000次的字符串操作，三者速度上的差距是惊人的。在确切知道是不变字符串的情况下，还是用String最好，因为Java语言中String采用了享元模式（Flyweight），在JVM中只存在一份相同的String 。在确定循环次数时，尽量使用for循环，在循环嵌套中尽量将大循环放到外层，小循环放到内层，这点与VB等语言恰好相反，经过代码测试的，不过还不知道原因。总之，在VB等微软系列语言中用小套大会快很多，而在JAVA中用大套小会快很多！当然这是指两者能互相替换的时候，比如二维数组遍觅等！**

**如果字符串特别长，采用charAt逐一获取特定位置的字符是非常耗时的。因为每次获取制定索引位置的字符都要引起新的检索过程，更好的办法是将字符串通过调用toCharArray方法转换成字符数组，然后通过数组索引值获得指定位置的字符。**

**对于boolean值，避免不必要的等式判断将一个boolean值与一个true比较是一个恒等操作(直接返回该boolean变量的值). 移走对于boolean的不必要操作至少会带来2个好处： 1)代码执行的更快 (生成的字节码少了5个字节)； 2)代码也会更加干净。**

**关于使用SESSION问题上，尽量不要将大对象放到HttpSession或其他须序列化的对象中，并注意及时清空Session。**