当面试官问你，ArrayList,List和LinkedList的区别的时候，他并不是仅仅想问你他们的用法，而是想透过这道题目，来了解你对日常工作中经常用到的东西有没有稍微深入的了解，深入到什么程度，有没有做到学以致用，把它们和数据结构紧密联系起来。通过这道小小的面试题，可以看出一个人对技术的专研大概程度。

一般，你这样回答，就会得到很好的分数。

    1.List是接口类，ArrayList和LinkedList是List的实现类。

    2.ArrayList是动态数组（顺序表）的数据结构。顺序表的存储地址是连续的，所以在查找比较快，但是在插入和删除时，由于需要把其它的元素顺序向后移动（或向前移动），所以比较熬时。

    3.LinkedList是链表的数据结构。链表的存储地址是不连续的，每个存储地址通过指针指向，在查找时需要进行通过指针遍历元素，所以在查找时比较慢。由于链表插入时不需移动其它元素，所以在插入和删除时比较快。

    仅仅是几句话，就把它们之间的最大不同道出，体现了你对List其实有着更深层的关注，并且知道数据结构层面的特点，面试官会感觉你不是浑浑噩噩在工作的人，平时会比较注意积累和思考。其实还有更深层次含义，就是一旦在工作中遇到问题，他会相信你能凭借这些见微知著的技术掌握很快找到原因并解决问题。

## 1、ArrayList的大小是如何自动增加的？你能分享一下你的代码吗？

这是最有技巧性的的一个问题，大多数人都无法回答。事实上，当有人试图在arraylist中增加一个对象的时候，Java会去检查arraylist，以确保已存在的数组中有足够的容量来存储这个新的对象。如果没有足够容量的话，那么就会新建一个长度更长的数组，旧的数组就会使用Arrays.copyOf方法被复制到新的数组中去，现有的数组引用指向了新的数组。看如下的代码段（摘自[GrepCode.com中的Java ArrayList Code](http://grepcode.com/file/repository.grepcode.com/java/root/jdk/openjdk/6-b14/java/util/ArrayList.java)）：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | //ArrayList Add方法：  public boolean add(E e){      ensureCapacity(size+1); //Increment modCount!!      elementData[size++] = e;      return true;  }    //ensureCapacity方法：处理ArrayList的大小  public void ensureCapacity(int minCapacity) {      modCount++;      int oldCapacity = elementData.length;      if (minCapacity > oldCapacity) {      Object oldData[] = elementData;      int newCapacity = (oldCapacity \* 3)/2 + 1;      if (newCapacity < minCapacity)          newCapacity = minCapacity;      // minCapacity is usually close to size, so this is a win:      elementData = Arrays.copyOf(elementData, newCapacity);      }  } |

请注意这样一个情况：新建了一个数组；旧数组的对象被复制到了新的数组中，并且现有的数组指向新的数组。

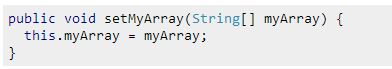
## 2、什么情况下你会使用ArrayList？什么时候你会选择LinkedList？

这又是一个大多数面试者都会困惑的问题。多数情况下，当你遇到访问元素比插入或者是删除元素更加频繁的时候，你应该使用ArrayList。另外一方面，当你在某个特别的索引中，插入或者是删除元素更加频繁，或者你压根就不需要访问元素的时候，你会选择LinkedList。这里的主要原因是，在ArrayList中访问元素的最糟糕的时间复杂度是”1″，而在LinkedList中可能就是”n”了。在ArrayList中增加或者删除某个元素，通常会调用System.arraycopy方法，这是一种极为消耗资源的操作，因此，在频繁的插入或者是删除元素的情况下，LinkedList的性能会更加好一点。

## 3、当传递ArrayList到某个方法中，或者某个方法返回ArrayList，什么时候要考虑安全隐患？如何修复安全违规这个问题呢？

当array被当做参数传递到某个方法中，如果array在没有被复制的情况下直接被分配给了成员变量，那么就可能发生这种情况，即当原始的数组被调用的方法改变的时候，传递到这个方法中的数组也会改变。下面的这段代码展示的就是安全违规以及如何修复这个问题。

ArrayList被直接赋给成员变量——安全隐患：

[](http://www.importnew.com/9928.html/1-array-stored-directly-considered-as-security-violation)

修复这个安全隐患：

# [http://www.importnew.com/wp-content/uploads/2014/03/81523f6d0a66f53529ccc7ce4810350d.jpg](http://www.importnew.com/9928.html/2-copy-the-array-as-a-fix-for-the-security-violation)

## 4、如何复制某个ArrayList到另一个ArrayList中去？写出你的代码？

下面就是把某个ArrayList复制到另一个ArrayList中去的几种技术：

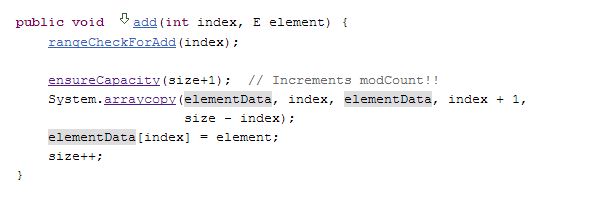
1. 使用clone()方法，比如ArrayList newArray = oldArray.clone();
2. 使用ArrayList构造方法，比如：ArrayList myObject = new ArrayList(myTempObject);
3. 使用Collection的copy方法。

注意1和2是浅拷贝(shallow copy)。

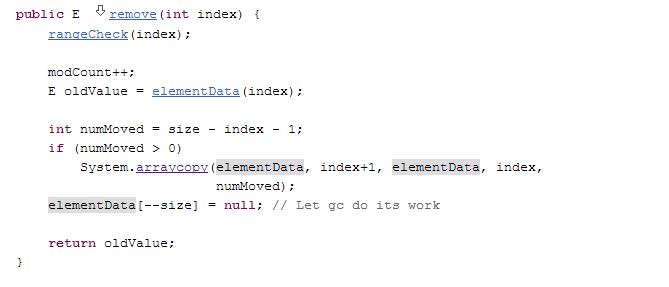
## 5、在索引中ArrayList的增加或者删除某个对象的运行过程？效率很低吗？解释一下为什么？

在ArrayList中增加或者是删除元素，要调用System.arraycopy这种效率很低的操作，如果遇到了需要频繁插入或者是删除的时候，你可以选择其他的Java集合，比如LinkedList。看一下下面的代码：

在ArrayList的某个索引i处添加元素：

[](http://www.importnew.com/9928.html/3-adding-an-element-in-arraylist-at-index-i)

删除ArrayList的某个索引i处的元素：

[](http://www.importnew.com/9928.html/4-removing-an-element-in-an-arraylist-at-index-i)

# 面试题：请自己实现一个Arraylist

MyIterator.java 自定义迭代器

/\*\*

\* 自定义迭代器

\*

\*/

public interface MyIterator {

boolean hasNext();

Object next();

}

MyArraylist.java 自己实现的Arraylist

import java.util.Arrays;

public class MyArraylist {

private Object[] objs = new Object[10]; //定义数组，初始化长度是10

private int index = 0; //数组下标

/\*\*

\* 添加

\*@param o

\*/

public void add(Object o){

if (index == objs.length){

// 扩容

extend();

}

objs[index++] =o;

}

/\*\*

\* 根据下标获取对象

\* @param index 下标

\* @return

\*/

public Object get(int index){

return objs[index];

}

/\*\*

\* 计算元素个数

\* @return

\*/

public int size(){

int size = 0;

for (int i = 0; i < objs.length; i++) {

if(objs[i]!=null){

size++;

}

}

return size;

}

/\*\*

\* 清空

\*/

public void clear(){

for (int i = 0; i < objs.length; i++) {

objs[i] = null;

}

}

/\*\*

\* 判断是否包含

\* @param o

\* @return

\*/

public boolean contains(Object o){

for (int i = 0; i < objs.length; i++) {

if(o!=null&&o.equals(objs[i])){

return true;

}

}

return false;

}

/\*\*

\* 根据下标删除

\*/

public void remove(int index){

if (index<0 || index>= this.size()){

System.out.println("下标无效，无法删除。。。");

throw new IndexOutOfBoundsException();

}

for (int i = index; i < objs.length; i++) {

objs[i] = objs[i+1];

if (i==this.size()){

break;

}

}

}

/\*\*

\* 根据对象删除

\* @param o

\*/

public void remove(Object o){

int index = getIndexByObject(o);

remove(index);

}

/\*\*

\* 根据对象获取对象的下标

\* @return

\*/

public int getIndexByObject(Object o){

for (int i = 0; i < objs.length; i++) {

if(o!=null&&o.equals(objs[i])){

return i;

}

}

return -1;

}

/\*\*

\* 自定义迭代器

\*/

public MyIterator iterator(){

return new Iterator();

}

class Iterator implements MyIterator{

private int size = 0;

@Override

public boolean hasNext() {

if(objs[size]!=null) return true;

else return false;

}

@Override

public Object next() {

size++;

return objs[size-1];

}

}

/\*\*

\* 扩容

\*/

private void extend(){

Arrays.copyOf(objs,objs.length+1);

}

}

Test.java 测试代码

import java.util.Arrays;

public class MyArraylist {

private Object[] objs = new Object[10]; //定义数组，初始化长度是10

private int index = 0; //数组下标

/\*\*

\* 添加

\*@param o

\*/

public void add(Object o){

if (index == objs.length){

// 扩容

extend();

}

objs[index++] =o;

}

/\*\*

\* 根据下标获取对象

\* @param index 下标

\* @return

\*/

public Object get(int index){

return objs[index];

}

/\*\*

\* 计算元素个数

\* @return

\*/

public int size(){

int size = 0;

for (int i = 0; i < objs.length; i++) {

if(objs[i]!=null){

size++;

}

}

return size;

}

/\*\*

\* 清空

\*/

public void clear(){

for (int i = 0; i < objs.length; i++) {

objs[i] = null;

}

}

/\*\*

\* 判断是否包含

\* @param o

\* @return

\*/

public boolean contains(Object o){

for (int i = 0; i < objs.length; i++) {

if(o!=null&&o.equals(objs[i])){

return true;

}

}

return false;

}

/\*\*

\* 根据下标删除

\*/

public void remove(int index){

if (index<0 || index>= this.size()){

System.out.println("下标无效，无法删除。。。");

throw new IndexOutOfBoundsException();

}

for (int i = index; i < objs.length; i++) {

objs[i] = objs[i+1];

if (i==this.size()){

break;

}

}

}

/\*\*

\* 根据对象删除

\* @param o

\*/

public void remove(Object o){

int index = getIndexByObject(o);

remove(index);

}

/\*\*

\* 根据对象获取对象的下标

\* @return

\*/

public int getIndexByObject(Object o){

for (int i = 0; i < objs.length; i++) {

if(o!=null&&o.equals(objs[i])){

return i;

}

}

return -1;

}

/\*\*

\* 自定义迭代器

\*/

public MyIterator iterator(){

return new Iterator();

}

class Iterator implements MyIterator{

private int size = 0;

@Override

public boolean hasNext() {

if(objs[size]!=null) return true;

else return false;

}

@Override

public Object next() {

size++;

return objs[size-1];

}

}

/\*\*

\* 扩容

\*/

private void extend(){

Arrays.copyOf(objs,objs.length+1);

}

}