[java集合框架之ArrayList深度解析（四）](http://www.spring4all.com/article/1056)

**摘要：**

本章内容主要讲解ArrayList的剩余部分的方法，包括了清空，增加所有等，相当于是ArrayList的源码讲解部分的最后一章，所以还是按照前面章节的步骤开始吧！

**ArrayList(Collection<? extends E> c)**

*/\*\**

*\* Constructs a list containing the elements of the specified*

*\* collection, in the order they are returned by the collection's*

*\* iterator.*

*\**

*\* @param c the collection whose elements are to be placed into this list*

*\* @throws NullPointerException if the specified collection is null*

*\*/*

public ArrayList(Collection<? extends E> c) {

elementData = c.toArray();

if ((size = elementData.length) != 0) {

*// c.toArray might (incorrectly) not return Object[] (see 6260652)*

if (elementData.getClass() != Object[].class)

elementData = Arrays.copyOf(elementData, size, Object[].class);

} else {

*// replace with empty array.*

this.elementData = EMPTY\_ELEMENTDATA;

}

}

首先讲Collection集合数据转化为数组elementData，长度赋值给size（相关分析见前面章节），如果长度为0的话就相当于无参构造器，最后判断具体类型，进行数组拷贝。

**trimToSize()**

*/\*\**

*\* Trims the capacity of this <tt>ArrayList</tt> instance to be the*

*\* list's current size. An application can use this operation to minimize*

*\* the storage of an <tt>ArrayList</tt> instance.*

*\*/*

public void trimToSize() {

modCount++;

if (size < elementData.length) {

elementData = (size == 0)

? EMPTY\_ELEMENTDATA

: Arrays.copyOf(elementData, size);

}

}

java.util.ArrayList.trimToSize()方法修整此ArrayList实例的是列表的当前大小的容量。应用程序可以使用此操作，以尽量减少一个ArrayList实例的存储。

ArrayList所说没有用的值并不是null，而是ArrayList每次增长会预申请多一点空间(如何扩容的请参考前面文章，绝对不是网上某些人说的1.5倍)，trimToSize() 的作用只是去掉预留元素位置，比如初始化值10，然后通过增加方法，多增加了1个元素，导致扩容为就是size=size+elementData>>1最后是15，但是实际占有11个元素，trimToSize()就是删除多余的没有使用的4个元素空间，改为只申请11,内存紧张的时候会用到.

**contains(Object o)**

**indexOf(Object o)**

**lastIndexOf(Object o)**

这三个方法放到一起讲解，因为它们直接有特别密切的联系，而又特别简单indexOf(Object o)从数组开头查找元素并且返回元素位于数组的下标，lastIndexOf(Object o)从数组的结尾逆向查找元素并且返回元素位于数组的下标。

*/\*\**

*\* Returns <tt>true</tt> if this list contains the specified element.*

*\* More formally, returns <tt>true</tt> if and only if this list contains*

*\* at least one element <tt>e</tt> such that*

*\* <tt>(o==null&nbsp;?&nbsp;e==null&nbsp;:&nbsp;o.equals(e))</tt>.*

*\**

*\* @param o element whose presence in this list is to be tested*

*\* @return <tt>true</tt> if this list contains the specified element*

*\*/*

public boolean contains(Object o) {

return indexOf(o) >= 0;

}

*/\*\**

*\* Returns the index of the first occurrence of the specified element*

*\* in this list, or -1 if this list does not contain the element.*

*\* More formally, returns the lowest index <tt>i</tt> such that*

*\* <tt>(o==null&nbsp;?&nbsp;get(i)==null&nbsp;:&nbsp;o.equals(get(i)))</tt>,*

*\* or -1 if there is no such index.*

*\*/*

public int indexOf(Object o) {

if (o == null) {

for (int i = 0; i < size; i++)

if (elementData[i]==null)

return i;

} else {

for (int i = 0; i < size; i++)

if (o.equals(elementData[i]))

return i;

}

return -1;

}

*/\*\**

*\* Returns the index of the last occurrence of the specified element*

*\* in this list, or -1 if this list does not contain the element.*

*\* More formally, returns the highest index <tt>i</tt> such that*

*\* <tt>(o==null&nbsp;?&nbsp;get(i)==null&nbsp;:&nbsp;o.equals(get(i)))</tt>,*

*\* or -1 if there is no such index.*

*\*/*

public int lastIndexOf(Object o) {

if (o == null) {

for (int i = size-1; i >= 0; i--)

if (elementData[i]==null)

return i;

} else {

for (int i = size-1; i >= 0; i--)

if (o.equals(elementData[i]))

return i;

}

return -1;

}

**clone()**

*/\*\**

*\* Returns a shallow copy of this <tt>ArrayList</tt> instance. (The*

*\* elements themselves are not copied.)*

*\**

*\* @return a clone of this <tt>ArrayList</tt> instance*

*\*/*

public Object clone() {

try {

ArrayList<?> v = (ArrayList<?>) super.clone();

v.elementData = Arrays.copyOf(elementData, size);

v.modCount = 0;

return v;

} catch (CloneNotSupportedException e) {

*// this shouldn't happen, since we are Cloneable*

throw new InternalError(e);

}

}

返回一个Object对象，所以在使用此方法的时候要强制转换。  
ArrayList的本质是维护了一个Object的数组，所以克隆也是通过数组的复制实现的，属于浅复制。  
编程人员经常误用各个集合类提供的拷贝构造函数作为克隆List，Set，ArrayList，HashSet或者其他集合实现的方法。需要记住的是，Java集合的拷贝构造函数只提供浅拷贝而不是深拷贝。这意味着存储在原始List和克隆List中的对象是相同的，指向Java堆内存中相同的位置。增加了这个误解的原因之一是对于不可变对象集合的浅克隆。由于不可变性，即使两个集合指向相同的对象是可以的。字符串池包含的字符串就是这种情况，更改一个不会影响到另一个。

不可变类：所谓的不可变类是指这个类的实例一旦创建完成后，就不能改变其成员变量值。如JDK内部自带的很多不可变类：Interger、Long和String等。  
可变类：相对于不可变类，可变类创建实例后可以改变其成员变量值，开发中创建的大部分类都属于可变类。

**一、浅度克隆**

浅度克隆对于要克隆的对象，对于其基本数据类型的属性，复制一份给新产生的对象，对于非基本数据类型的属性，仅仅复制一份引用给新产生的对象，即新产生的对象和原始对象中的非基本数据类型的属性都指向的是同一个对象。

**二、深度克隆**

在浅度克隆的基础上，对于要克隆的对象中的非基本数据类型的属性对应的类，也实现克隆，这样对于非基本数据类型的属性，复制的不是一份引用，即新产生的对象和原始对象中的非基本数据类型的属性指向的不是同一个对象

**小结：**

希望集合框架的clone()的各种方法都去尝试一下，包括了浅克隆和深克隆，这个地方就不做过多代码的示例了，毕竟不是该篇文章的重点，有机会的话会和大家一起分享它的奥秘！

**addAll(Collection<? extends E> c)，addAll(int index, Collection<? extends E> c)** 都是基于数组的拷贝，就不分析了特别的简单。最后分析一个比较有趣的方法，相信大家都知道，但是用得很少。。。。。

*/\*\**

*\* Removes from this list all of the elements whose index is between*

*\* {@code fromIndex}, inclusive, and {@code toIndex}, exclusive.*

*\* Shifts any succeeding elements to the left (reduces their index).*

*\* This call shortens the list by {@code (toIndex - fromIndex)} elements.*

*\* (If {@code toIndex==fromIndex}, this operation has no effect.)*

*\**

*\* @throws IndexOutOfBoundsException if {@code fromIndex} or*

*\* {@code toIndex} is out of range*

*\* ({@code fromIndex < 0 ||*

*\* fromIndex >= size() ||*

*\* toIndex > size() ||*

*\* toIndex < fromIndex})*

*\*/*

protected void removeRange(int fromIndex, int toIndex) {

modCount++;

int numMoved = size - toIndex;

System.arraycopy(elementData, toIndex, elementData, fromIndex,

numMoved);

*// clear to let GC do its work*

int newSize = size - (toIndex-fromIndex);

for (int i = newSize; i < size; i++) {

elementData[i] = null;

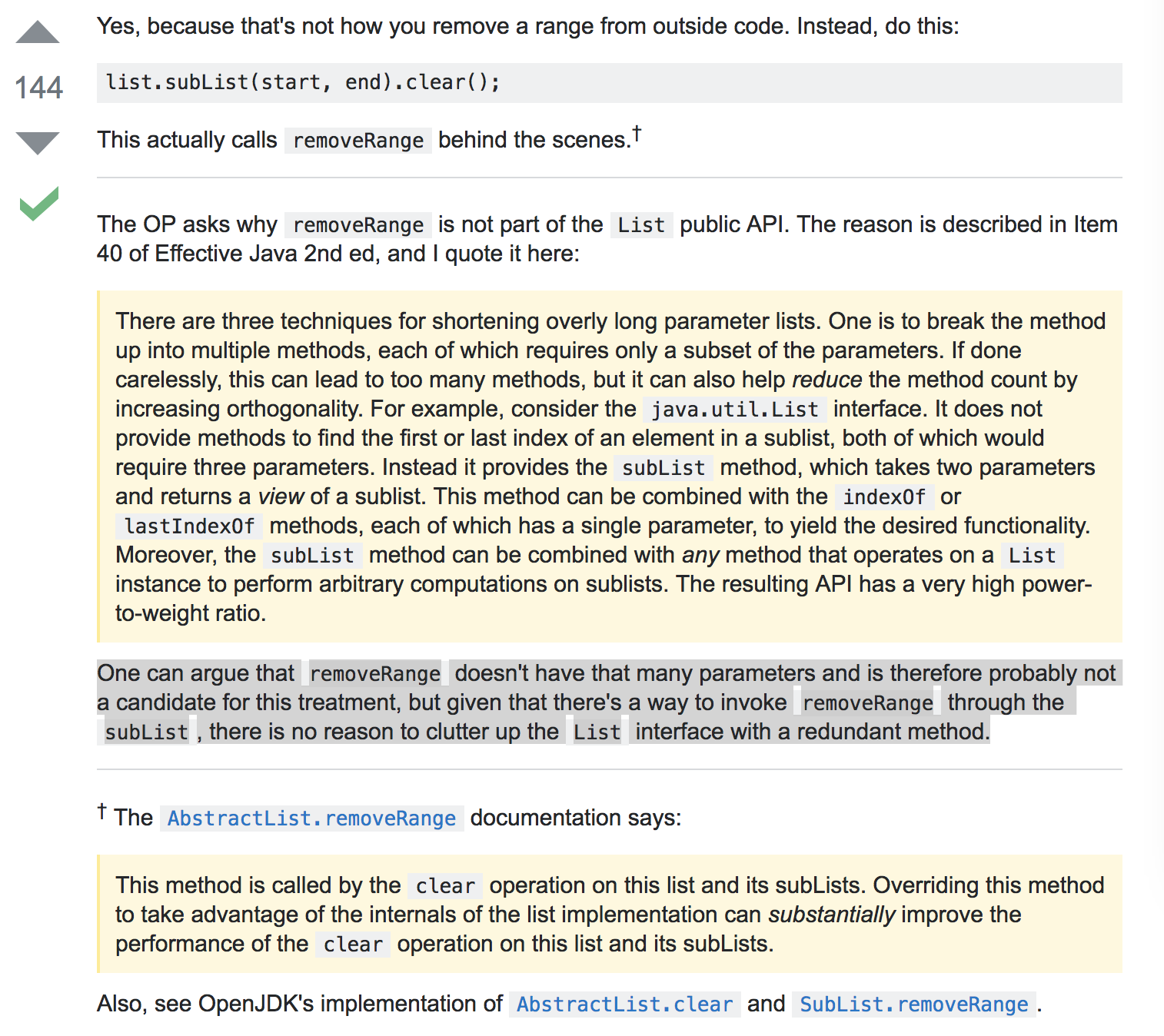
}

size = newSize;

}

执行过程是将elementData从toIndex位置开始的元素向前移动到fromIndex，然后将toIndex位置之后的元素全部置空顺便修改size。

**重点：**

这个方法是protected，及受保护的方法，为什么这个方法被定义为protected呢？为什么这个方法不暴露给用户使用呢？  
个人能力有限，只能找到网上这样的晦涩的答案

**译文：**  
有三种缩短过长参数列表的技巧。 一种是将方法分解成多种方法，每种方法只需要一部分参数。 如果不小心做到了，这会导致太多的方法，但它也可以通过增加正交性来帮助减少方法数量。 例如，考虑java.util.List接口。 它没有提供方法来查找子列表中元素的第一个或最后一个索引，这两个索引都需要三个参数。 相反，它提供了subList方法，它接受两个参数并返回一个子列表视图。 该方法可以与indexOf或lastIndexOf方法结合使用，每个方法都有一个参数，以产生所需的功能。 而且，subList方法可以与在List实例上运行的任何方法组合，以对子列表执行任意计算。 由此产生的API具有非常高的功率重量比。

[参考链接](https://stackoverflow.com/questions/2289183/why-is-javas-abstractlists-removerange-method-protected)

**总结：**

ArrayList部分的方法实在太博大精深了，时间和精力有限，只能分析到这个地方了，其他的方法不太常用的就没有提出来分析，可能遗漏的是移除所有removeAll(Collection<?> c)和批量移除batchRemove(Collection<?> c, boolean complement)这2个比较常用的方法吧，如果大家有兴趣的话可以自行去分析，其实分析集合框架对自己的能力提升是非常有帮助的，可以学习大神们的思维和写法，java中的集合框架毫无疑问也堪称是一门艺术！

**参考文献**

[Collections Framework Overview](https://docs.oracle.com/javase/6/docs/technotes/guides/collections/overview.html)  
[The For-Each Loop](https://docs.oracle.com/javase/1.5.0/docs/guide/language/foreach.html)

**参考本系列文章**

[Java 集合框架之概览](http://www.spring4all.com/article/1042)  
[java集合框架之ArrayList深度解析（一）](http://www.spring4all.com/article/1045)  
[java集合框架之ArrayList深度解析（二）](http://www.spring4all.com/article/1050)  
[java集合框架之ArrayList深度解析（三）](http://www.spring4all.com/article/1055)  
[java集合框架之ArrayList深度解析（四）](http://www.spring4all.com/article/1056)