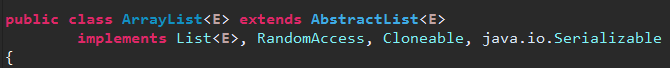
第1部分 ArrayList介绍

ArrayList简介

ArrayList 是一个**数组队列**，相当于 **动态数组**。与Java中的数组相比，它的容量能动态增长。它继承于AbstractList，实现了List, RandomAccess, Cloneable, java.io.Serializable这些接口。



ArrayList *继承了AbstractList，实现了List*。它是一个数组队列，提供了相关的添加、删除、修改、遍历等功能。  
ArrayList *实现了RandmoAccess接口，即提供了随机访问功能。*RandmoAccess是java中用来被List实现，为List提供快速访问功能的。在ArrayList中，我们即可以通过元素的序号快速获取元素对象；这就是快速随机访问。稍后，我们会比较ArrayList的“快速随机访问”和“通过Iterator迭代器访问”的效率。

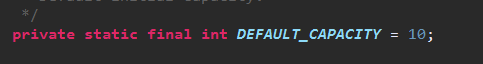
ArrayList 实现了Cloneable接口，即覆盖了函数clone()，能被克隆。

ArrayList 实现java.io.Serializable接口，这意味着ArrayList支持序列化，能通过序列化去传输。

和Vector不同，**ArrayList中的操作不是线程安全的**！所以，建议在单线程中才使用ArrayList，而在多线程中可以选择Vector或者CopyOnWriteArrayList。

ArrayList构造函数

ArrayList默认容量是10：



[复制代码](javascript:void(0);)

// 默认构造函数

ArrayList()

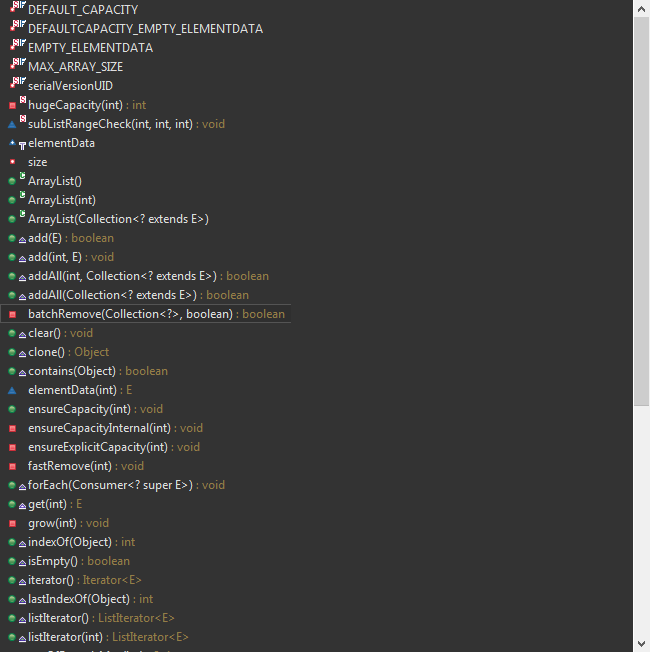
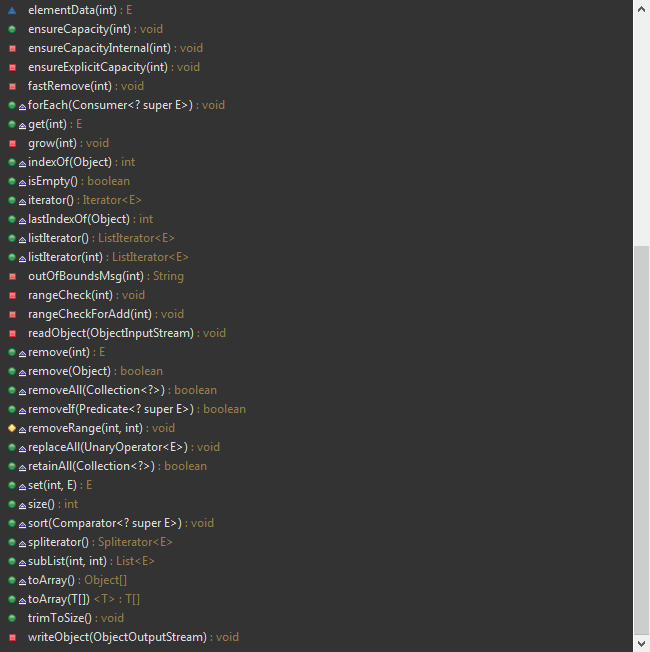
// capacity是ArrayList的默认容量大小。当由于增加数据导致容量不足时，容量会添加上一次容量大小的一半。

ArrayList(int capacity)

// 创建一个包含collection的ArrayList

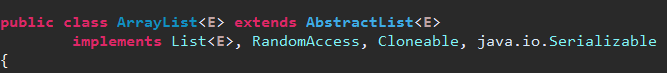
ArrayList(Collection<? extends E> collection)

[复制代码](javascript:void(0);)

ArrayList的API

第2部分 ArrayList数据结构

ArrayList的继承关系



















[复制代码](javascript:void(0);)

java.lang.Object

↳ java.util.AbstractCollection<E>

↳ java.util.AbstractList<E>

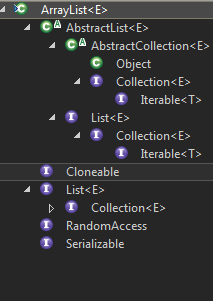
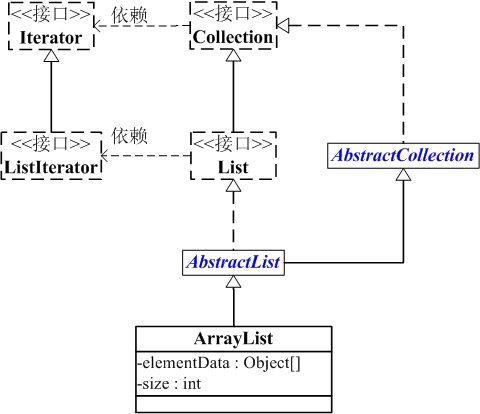
↳ java.util.ArrayList<E>

public class ArrayList<E> extends AbstractList<E>

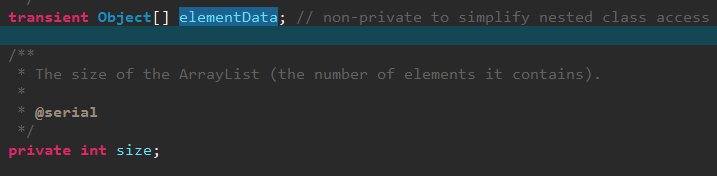
implements List<E>, RandomAccess, Cloneable, java.io.Serializable {}

[复制代码](javascript:void(0);)

ArrayList与Collection关系如下图：

[](https://images0.cnblogs.com/blog/497634/201401/272343457973489.jpg)

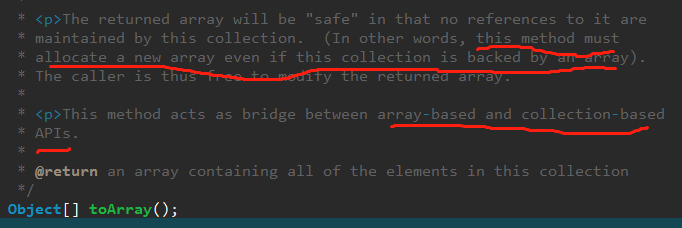
ArrayList包含了两个重要的对象：elementData 和 size。



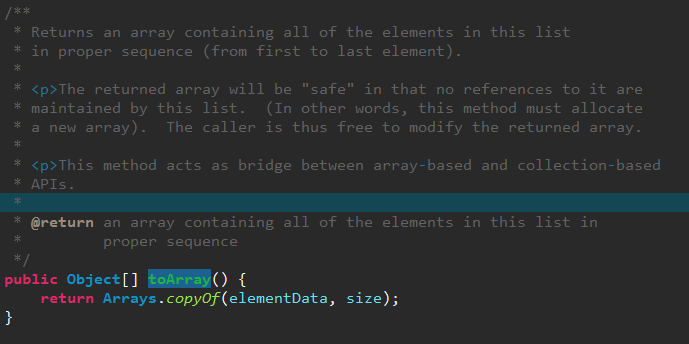
(01) elementData 是"Object[]类型的数组"，它保存了添加到ArrayList中的元素。实际上，elementData是个动态数组，我们能通过构造函数 ArrayList(int initialCapacity)来执行它的初始容量为initialCapacity；如果通过不含参数的构造函数ArrayList()来创建ArrayList，则elementData的容量默认是10。elementData数组的大小会根据ArrayList容量的增长而动态的增长，具体的增长方式，请参考源码分析中的ensureCapacity()函数。

附注，object的toarray函数与ensureCapacity（）函数：

下图为接口中的原始方法：

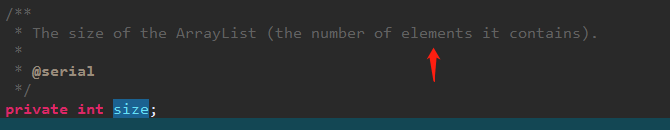


下图为ArrayList重写的toArray方法：

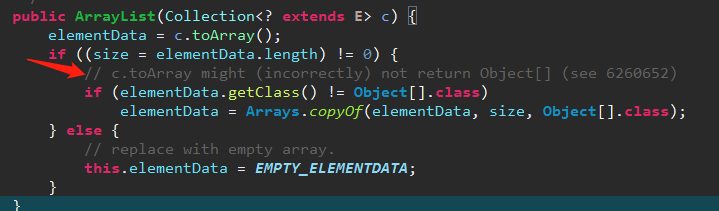




(02) size 则是动态数组的实际大小。

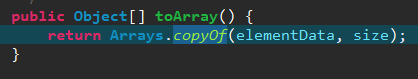


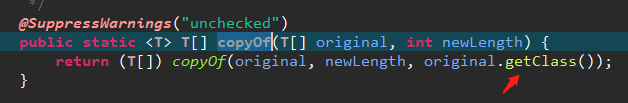
附注2：构造函数的问题：

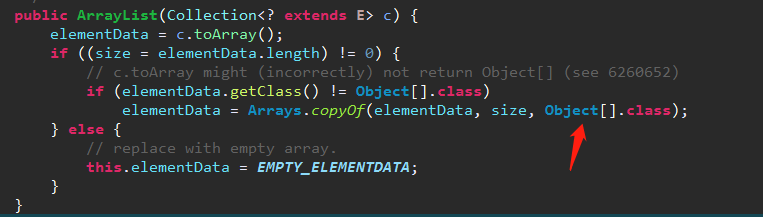


为什么要if呢？

首先看一下上面的两个toArray方法，构造函数中使用的是ArrayList中重写的方法，其getClass返回的是数组中真实的元素类型；而ArrayList的toArray方法调用的copyof方法使用的是original.class,故需要if一下，使用Object.class,因为elementdata是object[],懂了吗？



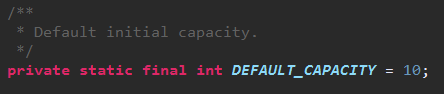


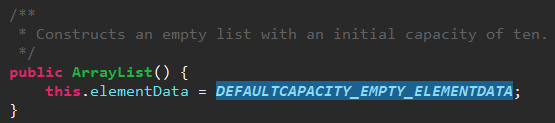


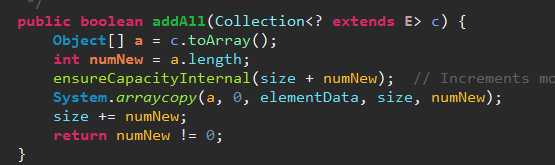
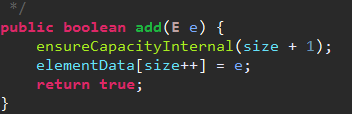
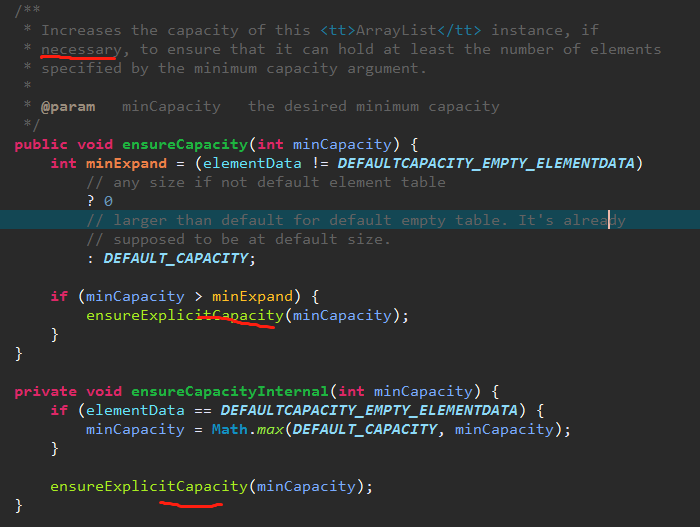
第3部分 ArrayList源码解析(基于JDK1.6.0\_45)

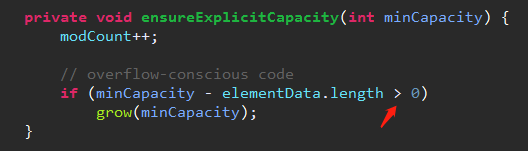
为了更了解ArrayList的原理，下面对ArrayList源码代码作出分析。ArrayList是通过数组实现的，源码比较容易理解。

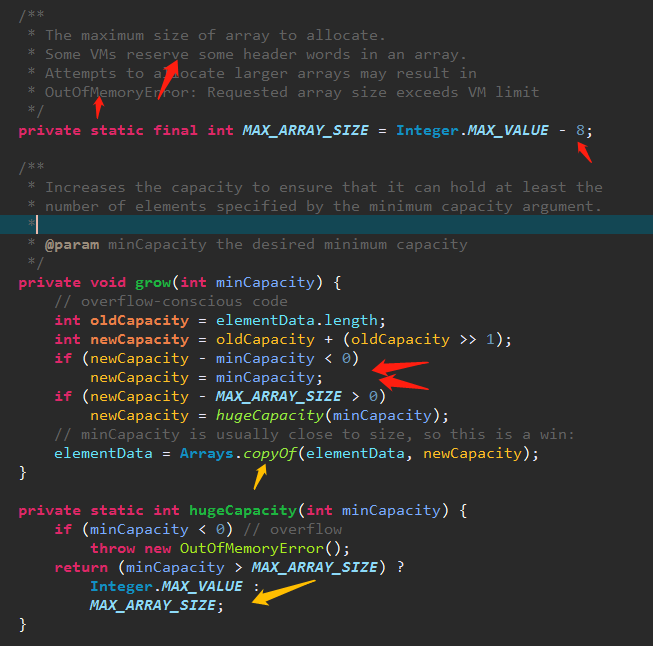
**总结**：  
(01) ArrayList 实际上是**通过一个数组去保存数据的**。当我们构造ArrayList时；若使用默认构造函数，则ArrayList的**默认容量大小是10**。



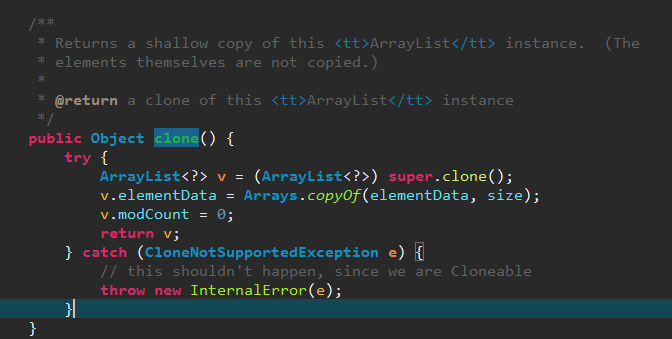
  
(02) 当ArrayList容量不足以容纳全部元素时，ArrayList会重新设置容量：**新的容量=“(原始容量x3)/2”**。每次添加元素时，都会调用ensurecapacity

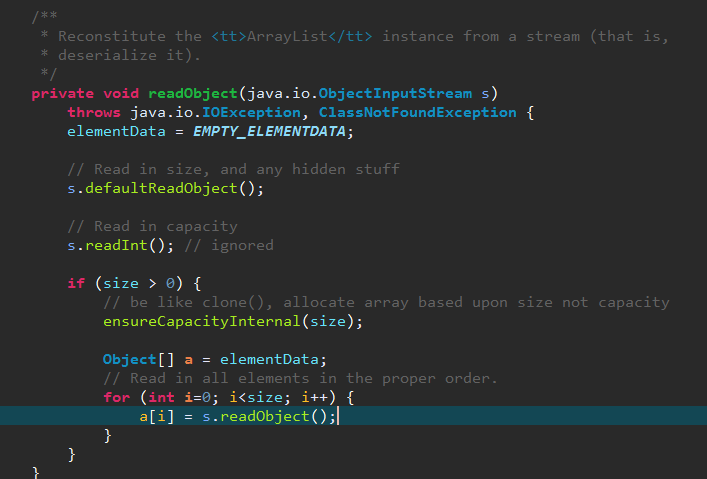


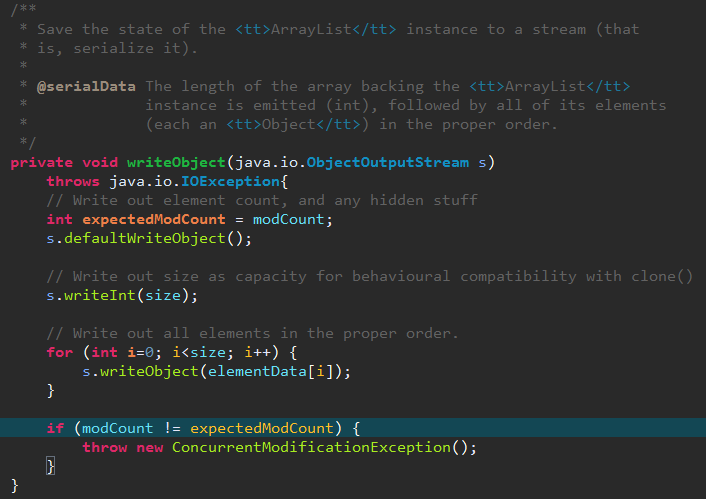




(03) ArrayList的克隆函数，即是将全部元素克隆到一个数组中。

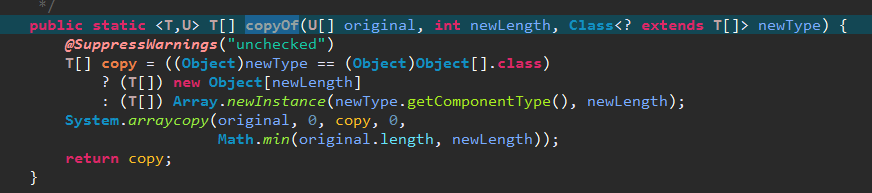
  
(04) ArrayList实现java.io.Serializable的方式。当写入到输出流时，先写入“容量”，再依次写入“每一个元素”；当读出输入流时，先读取“容量”，再依次读取“每一个元素”。





附注copyof：

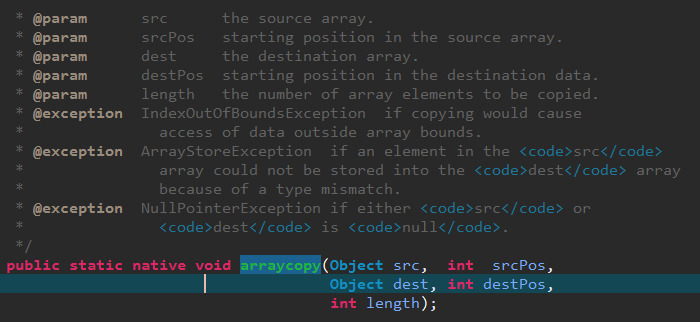
无论是copy还是clone还是arryList中的add，都是使用的System.arraycopy本地方法；



附注：核心思想：

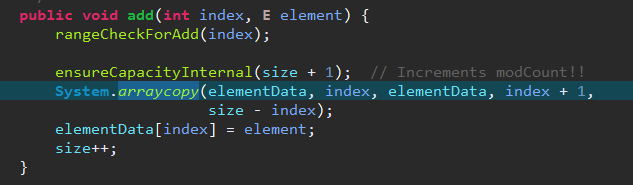
关于copyof的基本上都是用了system.arraycopy本地方法；

上述的ensure方法都会在grow中调用copyof方法，copyof方法有两种重载，参数较少的直接调用较多的那种，较多的会调用system.arraycopy方法，这是一个本地方法，有四个参数，而且add、addall方法也会调用该方法：

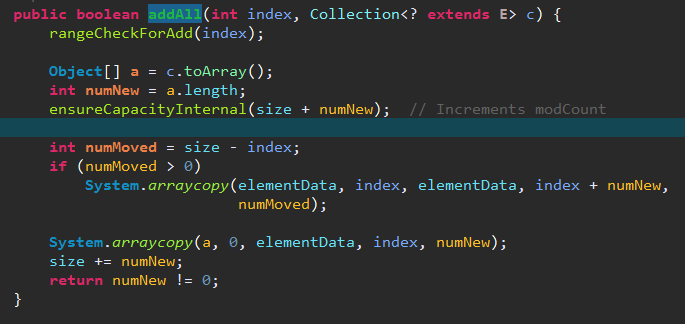


附注：toArray以及copyof调用链：

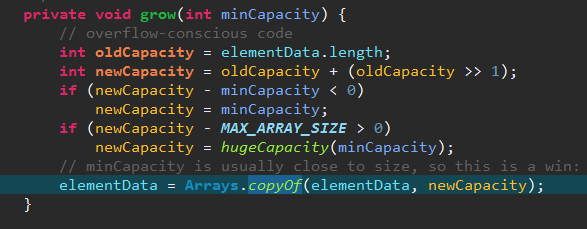
1）

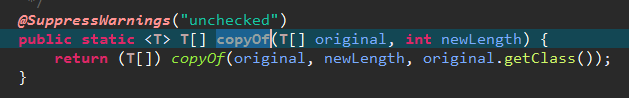


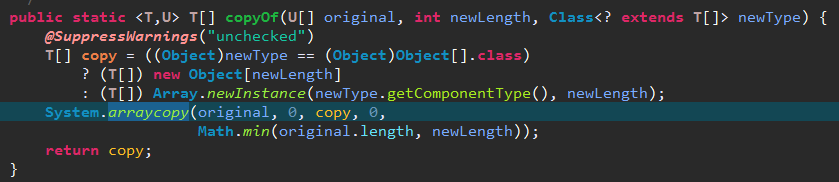
2）



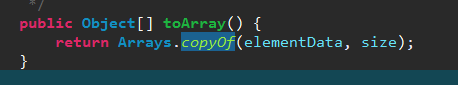
3）







4）



第4部分 ArrayList遍历方式

ArrayList支持3种遍历方式

(01) 第一种，**通过迭代器遍历**。即通过Iterator去遍历。

Integer value = null;

Iterator iter = list.iterator();

while (iter.hasNext()) {

value = (Integer)iter.next();

}

(02) 第二种，**随机访问，通过索引值去遍历。**  
由于ArrayList实现了RandomAccess接口，它支持通过索引值去随机访问元素。

Integer value = null;

int size = list.size();

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

value = (Integer)list.get(i);

}

(03) 第三种，**for循环遍历**。如下：

Integer value = null;

for (Integer integ:list) {

value = integ;

}

 由此可见，**遍历ArrayList时，使用随机访问(即，通过索引序号访问)效率最高，而使用迭代器的效率最低！**

第5部分 toArray()异常

当我们调用ArrayList中的 toArray()，可能遇到过抛出“java.lang.ClassCastException”异常的情况。下面我们说说这是怎么回事。

ArrayList提供了2个toArray()函数：

Object[] toArray()

<T> T[] toArray(T[] contents)

调用 toArray() 函数会抛出“java.lang.ClassCastException”异常，但是调用 toArray(T[] contents) 能正常返回 T[]。

toArray() 会抛出异常是因为 toArray() 返回的是 Object[] 数组，将 Object[] 转换为其它类型(如，将Object[]转换为的Integer[])则会抛出“java.lang.ClassCastException”异常，因为**Java不支持向下转型**。具体的可以参考前面ArrayList.java的源码介绍部分的toArray()。  
解决该问题的办法是调用 <T> T[] toArray(T[] contents) ， 而不是 Object[] toArray()。

调用 toArray(T[] contents) 返回T[]的可以通过以下几种方式实现。

[复制代码](javascript:void(0);)

// toArray(T[] contents)调用方式一

public static Integer[] vectorToArray1(ArrayList<Integer> v) {

Integer[] newText = new Integer[v.size()];

v.toArray(newText);

return newText;

}

// toArray(T[] contents)调用方式二。**最常用！**

public static Integer[] vectorToArray2(ArrayList<Integer> v) {

Integer[] newText = (Integer[])v.toArray(new Integer[0]);

return newText;

}

// toArray(T[] contents)调用方式三

public static Integer[] vectorToArray3(ArrayList<Integer> v) {

Integer[] newText = new Integer[v.size()];

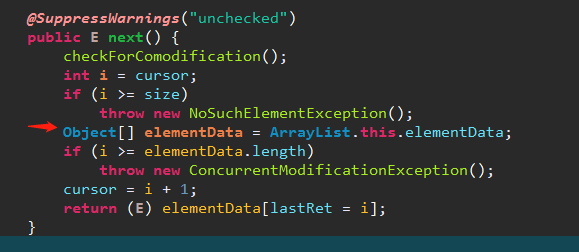
Integer[] newStrings = (Integer[])v.toArray(newText);

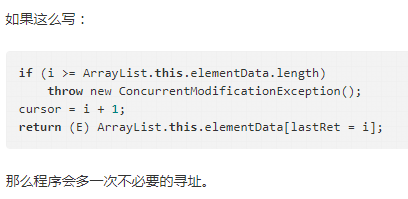
return newStrings;

}

附注：

1：为什么要这样写？



答案：