[java集合框架之ArrayList深度解析（三）](http://www.spring4all.com/article/1055)

**摘要：**

接下来将会从源码分析开始重点讲解ArrayList的增加和移除方法，以及部分数组在java虚拟机的底层实现原理，每一章讲解可能不会将所有的方法都讲完，也不能说完全讲透彻，所以尽可能的分几个章节来讲解ArrayList的源码分析。

**add(int index, E element)**

指定位置添加数据

*/\*\**

*\* Inserts the specified element at the specified position in this*

*\* list. Shifts the element currently at that position (if any) and*

*\* any subsequent elements to the right (adds one to their indices).*

*\**

*\* @param index index at which the specified element is to be inserted*

*\* @param element element to be inserted*

*\* @throws IndexOutOfBoundsException {@inheritDoc}*

*\*/*

public void add(int index, E element) {

rangeCheckForAdd(index);*//检查是否越界异常*

ensureCapacityInternal(size + 1); *// Increments modCount!!包括是否扩容*

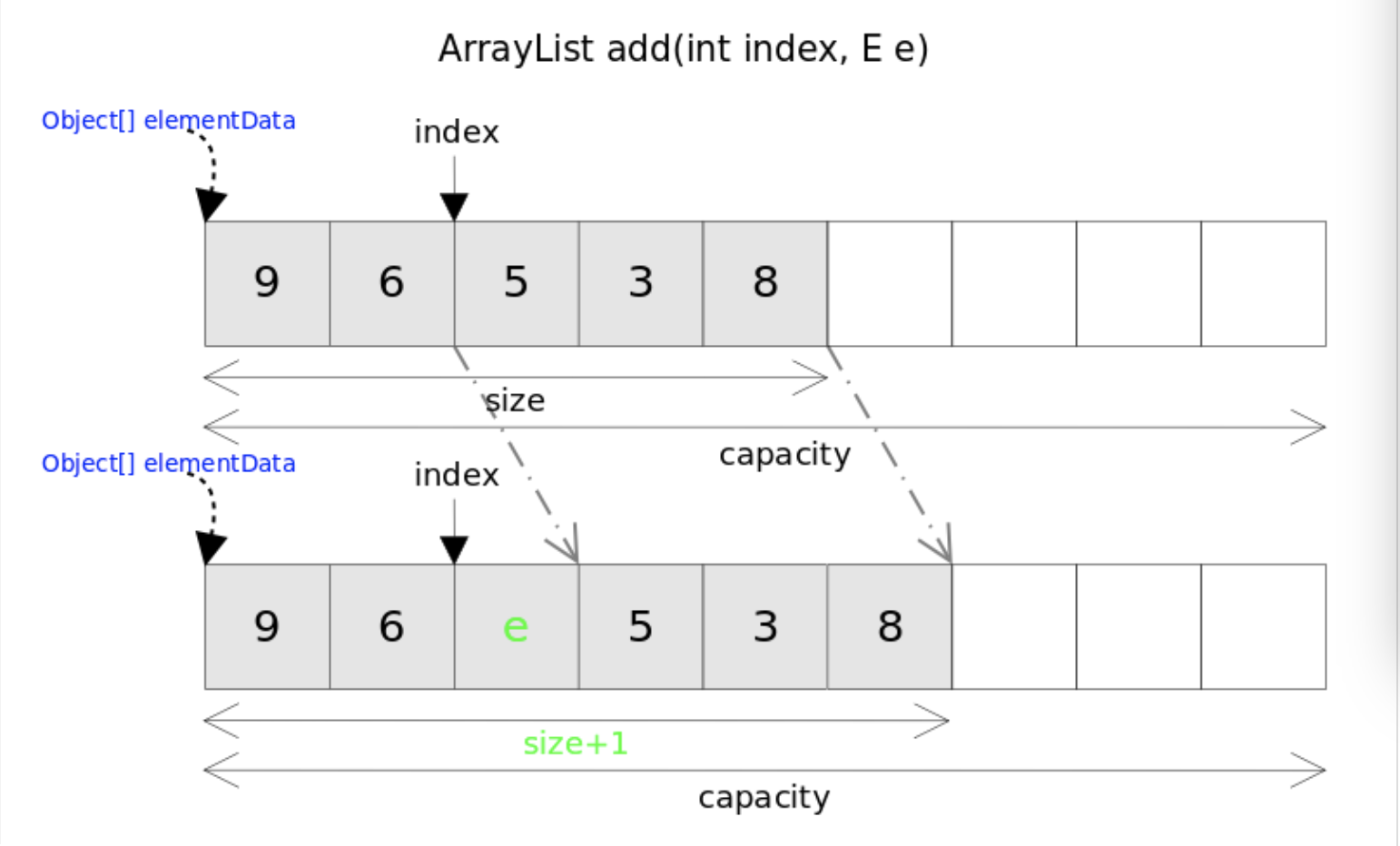
System.arraycopy(elementData, index, elementData, index + 1,

size - index);*//数组的拷贝，实际就是讲index位置的数据*

elementData[index] = element;

size++;

}



**rangeCheckForAdd(index)**

检查是否数组越界异常:size是一个成员变量初始值为0，也可以这样理解size的大小就是ArrayList中元素的实际个数，在指定位置添加数据是不能超过size大小的，可以结合上图示进行理解。

*/\*\**

*\* A version of rangeCheck used by add and addAll.*

*\*/*

private void rangeCheckForAdd(int index) {

if (index > size || index < 0)

throw new IndexOutOfBoundsException(outOfBoundsMsg(index));

}

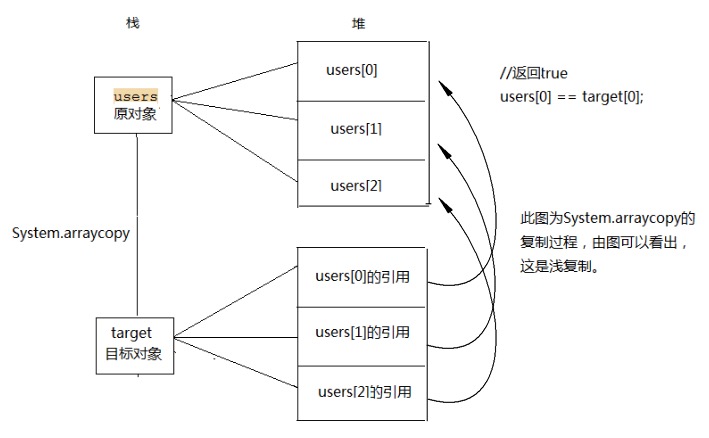
**System.arraycopy(Object src, int srcPos,Object dest, int destPos, int length）**

java.lang.System.arraycopy() 方法复制指定的源数组的数组，在指定的位置开始，到目标数组的指定位置。

下面是 System.arrayCopy的源代码声明 :

public static native void arraycopy(Object src, int srcPos, Object dest, int destPos,int length);

代码解释:  
Object src : 原数组  
int srcPos : 从元数据的起始位置开始  
Object dest : 目标数组  
int destPos : 目标数组的开始起始位置  
int length : 要copy的数组的长度

**图示：**对象复制的图示  
  
所以，得出的结论是，System.arraycopy() 在拷贝数组的时候，采用的使用潜复制，复制结果是一维的引用变量传递给副本的一维数组，修改副本时，会影响原来的数组。

**E remove(int index)**

指定位置移除数据

*/\*\**

*\* Removes the element at the specified position in this list.*

*\* Shifts any subsequent elements to the left (subtracts one from their*

*\* indices).*

*\**

*\* @param index the index of the element to be removed*

*\* @return the element that was removed from the list*

*\* @throws IndexOutOfBoundsException {@inheritDoc}*

*\*/*

public E remove(int index) {

rangeCheck(index);*//同上分析，检查数组是否越界*

modCount++;*//容量改变值增加，计数加1，更多详细内容请参考上一篇讲解*

E oldValue = elementData(index);*//获取删除位置上的值*

int numMoved = size - index - 1;*//进行数组拷贝时的指定的长度，*

if (numMoved > 0)

System.arraycopy(elementData, index+1, elementData, index,

numMoved);

elementData[--size] = null; *// clear to let GC do its work（参考《深入理解java虚拟机》或其它书籍）*

return oldValue;*//返回删除索引上的值*

}

**boolean remove(Object o)**

*/\*\**

*\* Removes the first occurrence of the specified element from this list,*

*\* if it is present. If the list does not contain the element, it is*

*\* unchanged. More formally, removes the element with the lowest index*

*\* <tt>i</tt> such that*

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

*\* (if such an element exists). Returns <tt>true</tt> if this list*

*\* contained the specified element (or equivalently, if this list*

*\* changed as a result of the call).*

*\**

*\* @param o element to be removed from this list, if present*

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

*\*/*

public boolean remove(Object o) {

if (o == null) {

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

if (elementData[index] == null) {

fastRemove(index);

return true;

}

} else {

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

if (o.equals(elementData[index])) {

fastRemove(index);

return true;

}

}

return false;

}

直接删除数据，核心方法在于fastRemove(index)，是不是很像指定索引的E remove(int index)，看下源码如下：

```java  
/\*

\* Private remove method that skips bounds checking and does not

\* return the value removed.

\*/

private void fastRemove(int index) {

modCount++;

int numMoved = size - index - 1;

if (numMoved > 0)

System.arraycopy(elementData, index+1, elementData, index,

numMoved);

elementData[--size] = null; *// clear to let GC do its work*

}

可以对比发现只是没有返回当前索引上的oldValue而已，然后注意一点儿根据上面的方法可以看出来（其实对java虚拟机规范熟悉的人就知道，ArrayList底层实现是数组）就是ArrayList是可以存放null值的。

**数组扩展知识**

Java语言中，数组的实现原理是什么？这个涉及到编译原理的问题，我只能说，这是一个编译规范。在规范中比如：int[]，中的int告诉计算机这是一个整型数据，[]告诉计算机这是一个连续存储的内存地址空间，简单点说一个连续数据的存储空间就是数组，数组只是一个名称！！当然我只是简略的这样说，实际上数组是很复杂的！  
数组在Java里是一种特殊类型，有别于普通的“类的实例”的对象。  
以HotSpot VM为例，答案是在数组对象的对象头里有一个\_length字段，记录数组长度。arraylength字节码的实现只要去读那个\_length字段即可。JVM 中数组对象是一种特殊的对象，它的Object Header 比普通对象多了一个word 来存储数组的长度，length 会编译成对应的字节码读取这个field 就可以了。

**总结：**

每一章讲得很少，很详细，因为我个人觉得循序渐进才是正确的学习的方法同时也能够增加理解，讲的不好的地方希望大家指出来，帮我纠正。

**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)