[java集合框架之ArrayList深度解析（一）](http://www.spring4all.com/article/1045)

**摘要：**

ArrayList实现了List接口，是顺序容器，即元素存放的数据与放进去的顺序相同，允许放入null元素，底层通过数组实现。除该类未实现同步外，其余跟Vector大致相同。每个ArrayList都有一个容量（capacity），表示底层数组的实际大小，容器内存储元素的个数不能多于当前容量。当向容器中添加元素时，如果容量不足，容器会自动增大底层数组的大小。前面已经提过，Java泛型只是编译器提供的语法糖，所以这里的数组是一个Object数组，以便能够容纳任何类型的对象。其中size(), isEmpty(), get(), set()方法均能在常数时间内完成，add()方法的时间开销跟插入位置有关，addAll()方法的时间开销跟添加元素的个数成正比。其余方法大都是线性时间。为追求效率，ArrayList没有实现同步（synchronized），如果需要多个线程并发访问，用户可以手动同步，也可使用Vector替代。

**创建ArrayList对象**

*//第一种方式采用默认长度的无参构造器创建ArrayList对象*

*/\*\**

*\* Shared empty array instance used for default sized empty instances. We*

*\* distinguish this from EMPTY\_ELEMENTDATA to know how much to inflate when*

*\* first element is added.*

*\*/*

private static final Object[] DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA = {};

*/\*\**

*\* The array buffer into which the elements of the ArrayList are stored.*

*\* The capacity of the ArrayList is the length of this array buffer. Any*

*\* empty ArrayList with elementData == DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA*

*\* will be expanded to DEFAULT\_CAPACITY when the first element is added.*

*\*/*

transient Object[] elementData; *// non-private to simplify nested class access*

*/\*\**

*\* Constructs an empty list with an initial capacity of ten.*

*\*/*

public ArrayList() {

this.elementData = DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA;

}

由以上源码分析很简单的得出一个结论，无参构造器给我们创建了一个长度为0的  
Object[]数组对象，这个Constructs an empty list with an initial capacity of ten.构造器的注释是怎么回事呢？为什么容量是10呢？借助以下方法解析来说明具体原因。。。。。。

**方法解析**

*/\*\**

*\* The size of the ArrayList (the number of elements it contains).*

*\**

*\* @serial*

*\*/*

private int size;

*/\*\**

*\* Default initial capacity.*

*\*/*

private static final int DEFAULT\_CAPACITY = 10;

*/\*\**

*\* Appends the specified element to the end of this list.*

*\**

*\* @param e element to be appended to this list*

*\* @return <tt>true</tt> (as specified by {@link Collection#add})*

*\*/*

public boolean add(E e) {

*// 加入元素前检查数组的容量是否足够,size是成员变量，代表ArrayList的中存在元素的个数，而不是ArrayList本身的长度*

ensureCapacityInternal(size + 1); *// Increments modCount!!*

elementData[size++] = e;

return true;

}

private void ensureCapacityInternal(int minCapacity) {

ensureExplicitCapacity(calculateCapacity(elementData, minCapacity));

}

private static int calculateCapacity(Object[] elementData, int minCapacity) {

*//在初始化ArrayList的时候就已经赋值*

if (elementData == DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA) {

return Math.max(DEFAULT\_CAPACITY, minCapacity);

}

return minCapacity;

}

因为默认的容量大小为10，所以这个地方会返回默认容量大小，对应了默认构造器的注释部分，默认容器大小为10，在方法ensureCapacityInternal(size + 1)中，每次都会先检查数组容量是否足够，并且都是先加1后比较，所以这个成员变量size会越来越大，导致后面的扩容机制发生。  
从上面的源码分析到计算了容量大小过后会进行是否进行扩容机制处理，源码如下：

private void ensureExplicitCapacity(int minCapacity) {

modCount++;*//我认为此处进行容量计数和size相同作用吧，本质原因不太清楚*

*// overflow-conscious code*

*//初始化无参构造器时minCapacity=10，并且不断增加，即第一次初始化过后直到size+1>10才会扩容*

if (minCapacity - elementData.length > 0)

grow(minCapacity);

}

初始化无参构造器的时候就会执行grow(minCapacity）方法，源码如下：

*/\*\**

*\* Increases the capacity to ensure that it can hold at least the*

*\* number of elements specified by the minimum capacity argument.*

*\**

*\* @param minCapacity the desired minimum capacity*

*\*/*

private void grow(int minCapacity) {

*// overflow-conscious code*

int oldCapacity = elementData.length;

int newCapacity = oldCapacity + (oldCapacity >> 1);

if (newCapacity - minCapacity < 0)

newCapacity = minCapacity;*//minCapacity为10*

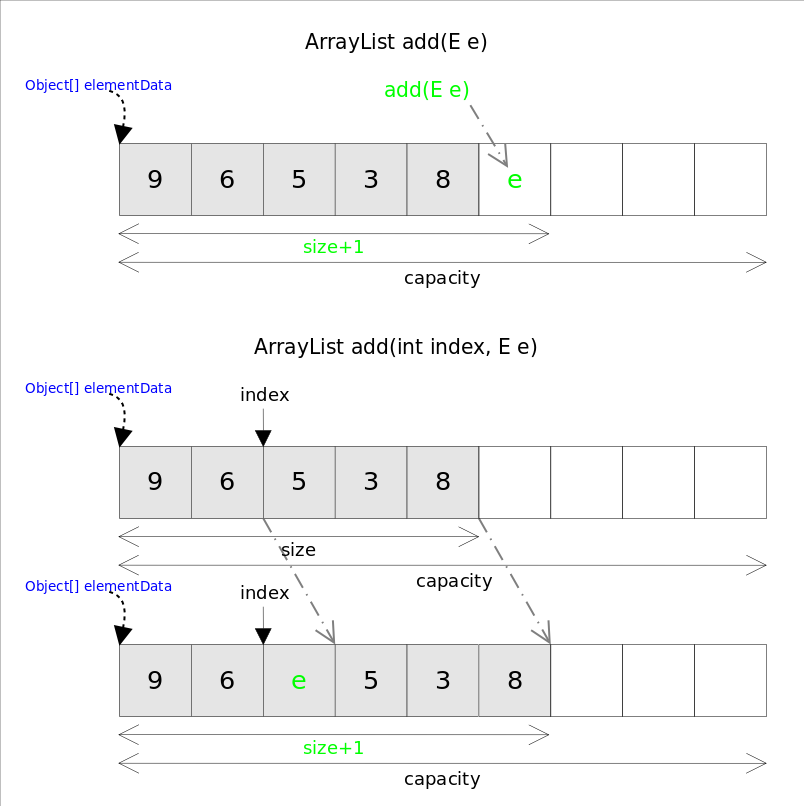
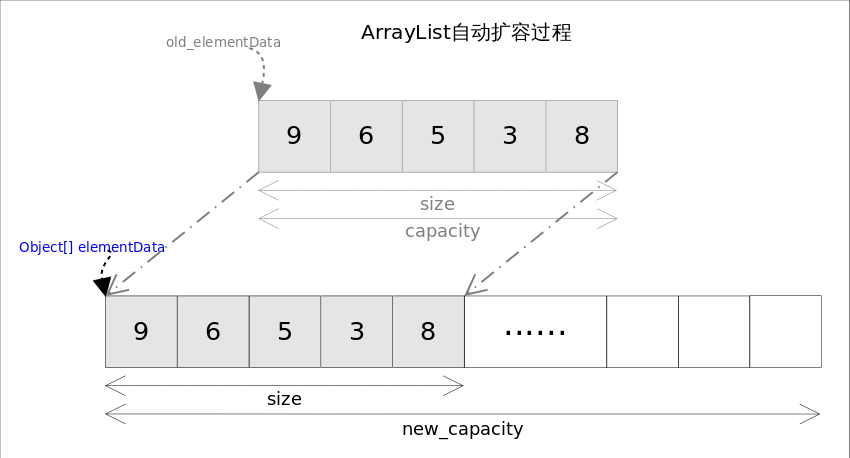
if (newCapacity - MAX\_ARRAY\_SIZE > 0)

newCapacity = hugeCapacity(minCapacity);

*// minCapacity is usually close to size, so this is a win:*

elementData = Arrays.copyOf(elementData, newCapacity);

}



**补充：**

--- 带符号右移运算符，它表示将运算符左边的运算对象，向右移动运算符右边指定的位数。如果是正数，在高位补零，如果是负数，则在高位补1；

初始化容量为10 二进制表示为1010，带符号右移1为表示为0101结果为5，所以扩容后容器大小为15，二进制表示为1111，如果继续扩容，将会增加0111，即增加7，扩容后大小为22，也不是人家说的进行1.5倍扩容。

**扩容机制核心源码**

*// Cloning*

*/\*\**

*\* Copies the specified array, truncating or padding with nulls (if necessary)*

*\* so the copy has the specified length. For all indices that are*

*\* valid in both the original array and the copy, the two arrays will*

*\* contain identical values. For any indices that are valid in the*

*\* copy but not the original, the copy will contain <tt>null</tt>.*

*\* Such indices will exist if and only if the specified length*

*\* is greater than that of the original array.*

*\* The resulting array is of exactly the same class as the original array.*

*\**

*\* @param <T> the class of the objects in the array*

*\* @param original the array to be copied*

*\* @param newLength the length of the copy to be returned*

*\* @return a copy of the original array, truncated or padded with nulls*

*\* to obtain the specified length*

*\* @throws NegativeArraySizeException if <tt>newLength</tt> is negative*

*\* @throws NullPointerException if <tt>original</tt> is null*

*\* @since 1.6*

*\*/*

@SuppressWarnings("unchecked")

public static <T> T[] copyOf(T[] original, int newLength) {

return (T[]) copyOf(original, newLength, original.getClass());

}

*/\*\**

*\* Copies the specified array, truncating or padding with nulls (if necessary)*

*\* so the copy has the specified length. For all indices that are*

*\* valid in both the original array and the copy, the two arrays will*

*\* contain identical values. For any indices that are valid in the*

*\* copy but not the original, the copy will contain <tt>null</tt>.*

*\* Such indices will exist if and only if the specified length*

*\* is greater than that of the original array.*

*\* The resulting array is of the class <tt>newType</tt>.*

*\**

*\* @param <U> the class of the objects in the original array*

*\* @param <T> the class of the objects in the returned array*

*\* @param original the array to be copied*

*\* @param newLength the length of the copy to be returned*

*\* @param newType the class of the copy to be returned*

*\* @return a copy of the original array, truncated or padded with nulls*

*\* to obtain the specified length*

*\* @throws NegativeArraySizeException if <tt>newLength</tt> is negative*

*\* @throws NullPointerException if <tt>original</tt> is null*

*\* @throws ArrayStoreException if an element copied from*

*\* <tt>original</tt> is not of a runtime type that can be stored in*

*\* an array of class <tt>newType</tt>*

*\* @since 1.6*

*\*/*

public static <T,U> T[] copyOf(U[] original, int newLength, Class<? extends T[]> newType) {

@SuppressWarnings("unchecked")

T[] copy = ((Object)newType == (Object)Object[].class)

? (T[]) new Object[newLength]

: (T[]) Array.newInstance(newType.getComponentType(), newLength);

System.arraycopy(original, 0, copy, 0,

Math.min(original.length, newLength));

return copy;

}

**小结：**

本文讲解了一下关于无参构造器创建ArrayList对象，以及add(E e)方法的原理，后续文章会继续讲解ArrayList的其它方法和有参构造器的源码分析，和大家从源码角度来理解java集合框架，方便以后的使用，最重要的是**万丈高楼平地起**，重视基础才是立于java的不败之地！

**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)