**ArrayList自动扩容解析**

ArrayList的列表对象实质上是存储在一个引用型数组里的，有人认为该数组有“自动增长机制”可以自动改变size大小。正式地说，该数组是无法改变 大小的，实际上它只是改变了该引用型数组的指向而已。下面，让我们来看看java是怎样实现ArrayList类的。

一、ArrayList类的实质

ArrayList底层采用Object类型的数组实现，当使用不带参数的构造方法生成ArrayList对象时， 实际上会在底层生成一个长度为10的Object类型数组。

首先，ArrayList定义了一个私有的未被序列化的数组elementData，用来存储ArrayList的对象列表（注意只定义未初始）：private transient Object[] elementData;

其次，以指定初始容量（Capacity）或把指定的Collection转换为引用型数组后实例化elementData数组；如果没有指定，则预置初始容量为10进行实例化。把私有数组预先实例化，然后通过copyOf方法覆盖原数组，是实现自动改变ArrayList的大小(size)的关键。ArrayList的构造方法源码如下：

//用指定的初始容量构造一个空列表。

public ArrayList(int initialCapacity) {

if (initialCapacity > 0) {

this.elementData = new Object[initialCapacity];

} else if (initialCapacity == 0) {

this.elementData = EMPTY\_ELEMENTDATA;

} else {

throw new IllegalArgumentException("Illegal Capacity: "+

initialCapacity);

}

}

二、ArrayList实现自动改变size机制

     为了实现这一机制，java引进了Capacity和size概念，以区别数组的length。为了保证用户增加新的列表对象，java设置了最小容量（minCapacity） ，通常情况上，它大于列表对象的数目，所以Capactiy虽然就是底层数组的长度（length），但是对于最终用户来讲，它是无意义的。而size存储着列表 对象的数量，才是最终用户所需要的。为了防止用户错误修改，这一属性被设置为privae的，不过可以通过size()获取。

下面，对ArrayList的初始以及其列表对象的增加和删除等三种情况下的size自动改变机制进行分析

初始Capacity和size值。 从上面给出的ArrayList构造方法源码中，我们不难看出Capacity初始值（initialCapacity）可以由用户直接指定或由用户指定的Collection集合存 储的对象数目确定，如果没有指定，系统默认为10。而size的被声明为int型变量，默认为0，当用户指定Collection创建ArrayList时，size值等于 initialCapacity。

add()方法 该方法的源码如下：

public boolean add(E e) {

ensureCapacityInternal(size + 1);

elementData[size++] = e;//添加对象时，自增size

return true;

}

方法中调用的ensureCapacityInternal主要用来调整容量，修改elementData数组的指向。其中涉及到3个方法的调用，其核心在于grow方法：

private void ensureCapacityInternal(int minCapacity) {

modCount++;//定义于ArrayList的父类AbstractList，用于存储结构修改次数

// overflow-conscious code

if (minCapacity - elementData.length > 0)

grow(minCapacity);

}

private void grow(int minCapacity) {

// overflow-conscious code

int oldCapacity = elementData.length;

int newCapacity = oldCapacity + (oldCapacity >> 1);//新容量扩大到原容量的1.5倍，右移一位相关于原数值除以2。

if (newCapacity - minCapacity < 0)

newCapacity = minCapacity;

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);

}

private static int hugeCapacity(int minCapacity) {

if (minCapacity < 0) // overflow

throw new OutOfMemoryError();

return (minCapacity > MAX\_ARRAY\_SIZE) ?

Integer.MAX\_VALUE :

MAX\_ARRAY\_SIZE;//MAX\_ARRAY\_SIZE和Integer.MAX\_VALUE为常量，详细请参阅下面的注解

}

通过以上代码，我们可知java自动增加ArrayList大小的思路是：向ArrayList添加对象时，原对象数目加1如果大于原底层数组长度，则以适当长度新 建一个原数组的拷贝，并修改原数组，指向这个新建数组。原数组自动抛弃（java垃圾回收机制会自动回收）。size则在向数组添加对象，自增1。

注解： //定义于该类的常量，用来分配数组的size最大值。一些 VMs在数组里保留字头，试图分配更大数组时可能导致OutOfMemoryError:被请求数组的 size超出VM界限。

private static final int MAX\_ARRAY\_SIZE = Integer.MAX\_VALUE - 8;

//在java.lang.Integer类中常量MIN\_VALUE、MAX\_VALUE如下：

public static final int MIN\_VALUE = 0x80000000;//整型取值区间下界：－2147483648

public static final int MAX\_VALUE = 0x7fffffff;//整型取值区间上界：2147483647

　　//在java.util.AbstractList中modCount定义如下：

　　protected transient int modCount = 0;

　 3. remove()方法 该重构方法其一源码如下（其它的就不累述了）：

public E remove(int index) {

rangeCheck(index);

modCount++;

E oldValue = elementData(index);

int numMoved = size - index - 1;

if (numMoved > 0)

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

numMoved);//将后面的列表对象前移

elementData[--size] = null; // 数组前移一位，size自减，空出来的位置置null，具体的对象的销毁由Junk收集器负责

return oldValue;

}

private void rangeCheck(int index) {//边界检查

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

throw new IndexOutOfBoundsException(outOfBoundsMsg(index));

}

E elementData(int index) {//获取指定index所在位置的对象

return (E) elementData[index];

}

通过remove()源码的学习，我们不难看出，其改变ArrayList大小的核心与add()方法相似，都是同数组拷贝。 另外，如果确有必要，用户也可以指定ArrayList实例的容量，可以有效的降低时间成本。它是通过调用ensureCapacityInternal来实现的，源代码 如下：

public void ensureCapacity(int minCapacity) {

if (minCapacity > 0)

ensureCapacityInternal(minCapacity);

}

因为size为private的，java给出方法来访问它：

public int size() {

checkForComodification();

return this.size;

}

综上所述，在用户向ArrayList追加对象时，Java总是要先计算容量（Capacity）是否适当，若容量不足则把原数组拷贝到以指定容量为长度创建的 新数组内，并对原数组变量重新赋值，指向新数组。在这同时，size进行自增1。在删除对象时，先使用拷贝方法把指定index后面的对象前移1位（如果 有的话），然后把空出来的位置置null，交给Junk收集器销毁，size自减1。