集合框架（三）ArrayList 源码分析

作者：alex18595752445
原文链接：https://ld246.com/article/1583331887518
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

ArrayList 源码分析

一、简介

ArrayList 是集合的一种实现，实现了接口 List，List 接口继承了 Collection 接口。Collection 是所集合类的父类。ArrayList 使用非常广泛，不论是数据库表查询，Excel 导入解析，还是网站数据爬取需要使用到，了解 ArrayList 原理及使用方法显得非常重要。

定义一个 ArrayList

```
//默认创建一个ArrayList集合
List<String> list = new ArrayList<>();
//创建一个初始化长度为100的ArrayList集合
List<String> initlist = new ArrayList<>(100);
//将其他类型的集合转为ArrayList
List<String> setList = new ArrayList<>(new HashSet());
```

我们读一下源码，看看定义 ArrayList 的过程到底做了什么？

```
public class ArrayList<E> extends AbstractList<E> implements List<E>, RandomAccess, Clone
ble, java.io.Serializable {
    /**
     * Default initial capacity.
     */
    private static final int DEFAULT_CAPACITY = 10;

    /**
     * Shared empty array instance used for empty instances.
     */
    private static final Object[] EMPTY_ELEMENTDATA = {};
```

```java
/**
 * 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

/**
 * The size of the ArrayList (the number of elements it contains).
 *
 * @serial
 */
private int size;

/**
 * Constructs an empty list with the specified initial capacity.
 *
 * @param  initialCapacity  the initial capacity of the list
 * @throws IllegalArgumentException if the specified initial capacity
 *         is negative
 */
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);
    }
}

/**
 * Constructs an empty list with an initial capacity of ten.
 */
public ArrayList() {
    this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA;
}

/**
 * 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;
        }
    }
}
```

其实源码里面已经很清晰了，ArrayList 非线程安全，底层是一个 Object[]，添加到 ArrayList 中的数保存在了 elementData 属性中。

- 当调用 `new ArrayList<>()` 时，将一个空数组{}赋值给了 elementData，这个时候集合的长度 siz 为默认长度 0；
- 当调用 `new ArrayList<>(100)` 时，根据传入的长度，new 一个 Object[100]赋值给 elementDa a，当然如果玩儿的话，传了一个 0，那么将一个空数组{}赋值给了 elementData；
- 当调用 new ArrayList<>(new HashSet())时，根据源码，我们可知，可以传递任何实现了 Collecti n 接口的类，将传递的集合调用 toArray()方法转为数组内赋值给 elementData；

注意 在传入集合的 ArrayList 的构造方法中，有这样一个判断

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

给出的注释是：c.toArray might (incorrectly) not return Object[] (see 6260652)，即调用 toArray 方法返回的不一定是 Object[]类型，查看 ArrayList 源码

```
public Object[] toArray() {    return Arrays.copyOf(elementData, size);}
```

我们发现返回的确实是 Object[]，那么为什么还会有这样的判断呢？

如果有一个类 CustomList 继承了 ArrayList，然后重写了 toArray()方法呢。

```
public class CustomList<E> extends ArrayList {
    @Override
    public Integer [] toArray() {
        return new Integer[]{1,2};
    };

    public static void main(String[] args) {
        Object[] elementData = new CustomList<Integer>().toArray();
        System.out.println(elementData.getClass());
        System.out.println(Object[].class);
        System.out.println(elementData.getClass() == Object[].class);
    }
```

执行结果：

```
class [Ljava.lang.Integer;
class [Ljava.lang.Object;
false
```

接着说，如果传入的集合类型和我们定义用来保存添加到集合中值的 Object[]类型不一致时，ArrayLis 做了什么处理？读源码看到，调用了 Arrays.copyOf(elementData, size, Object[].class);，继续往走

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

我们发现定义了一个新的数组，将原数组的数据拷贝到了新的数组中去。

二 ArrayList 常用方法

ArrayList 有很多常用方法，add，addAll，set，get，remove，size，isEmpty 等

首先定义了一个 ArrayList,

```
List<String> list = new ArrayList<>(10);
list.add('牛魔王');
list.add('蛟魔王');
...
list.add('美猴王');
```

Object[] elementData 中数据如下：

牛魔王	蛟魔王	鹏魔王	狮驼王	猕猴王	禺狨王	美猴王			

1. add(E element)

我们通过源码来看一下 add("白骨精")到底发生了什么

```
public boolean add(E e) {
    ensureCapacityInternal(size + 1);
    // Increments modCount!!
    elementData[size++] = e;
    return true;
}
```

首先通过 ensureCapacityInternal(size + 1) 来保证底层 Object[]数组有足够的空间存放添加的数据然后将添加的数据存放到数组对应的位置上，我们看一下是怎么保证数组有足够的空间？

<pre class="java" style="margin: 10px 0px; padding: 0px; white-space: pre-wrap; overflow-wrp: break-word; color: rgb(51, 51, 51); font-size: 13.3333px; font-style: normal; font-variant-ligaures: normal; font-variant-caps: normal; font-weight: 400; letter-spacing: normal; orphans: 2; txt-align: start; text-indent: 0px; text-transform: none; widows: 2; word-spacing: 0px; -webkit-txt-stroke-width: 0px; background-color: rgb(255, 255, 255); text-decoration-style: initial; text-

ecoration-color: initial;"><br class="Apple-interchange-newline"/></pre>

```
private void ensureCapacityInternal(int minCapacity) {
    if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
        minCapacity = Math.max(DEFAULT_CAPACITY, minCapacity);
    }
    ensureExplicitCapacity(minCapacity);
}


private void ensureExplicitCapacity(int minCapacity) {
    modCount++;
    // overflow-conscious code
    if (minCapacity - elementData.length > 0)
        grow(minCapacity);
}
```

这里首先确定了 Object[]足够存放添加数据的最小容量，然后通过 grow(int minCapacity) 来进行数扩容

<pre class="java" style="margin: 10px 0px; padding: 0px; white-space: pre-wrap; overflow-wrp: break-word; color: rgb(51, 51, 51); font-size: 13.3333px; font-style: normal; font-variant-ligaures: normal; font-variant-caps: normal; font-weight: 400; letter-spacing: normal; orphans: 2; txt-align: start; text-indent: 0px; text-transform: none; widows: 2; word-spacing: 0px; -webkit-txt-stroke-width: 0px; background-color: rgb(255, 255, 255); text-decoration-style: initial; text-ecoration-color: initial;"><br class="Apple-interchange-newline"/></pre>

```
private void grow(int minCapacity) {
    // overflow-conscious code
    int oldCapacity = elementData.length;
    int newCapacity = oldCapacity + (oldCapacity >> 1);
    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);
}
```

扩容规则为“**数组当前足够的最小容量 + （数组当前足够的最小容量 / 2）**”，即**数组当前足够的最容量 * 1.5**，当然有最大值的限制。

因为最开始定义了集合容量为 10，故而本次不会进行扩容，直接将第 8 个位置（从 0 开始，下标为）设置为“白骨精”，这时 Object[] elementData 中数据如下：

牛魔王	蛟魔王	鹏魔王	狮驼王	猕猴王	禺狨王	美猴王	白骨精		

还有和 add()类似的方法。空间扩容原理都是一样，如：

add("铁扇", 0); //将数组中的元素各自往后移动一位，再将“铁扇”放到第一个位置上；

铁扇	牛魔王	蛟魔王	鹏魔王	狮驼王	猕猴王	禺狨王	美猴王	白骨精		

addAll(list..七个葫芦娃); //将集合{七个葫芦娃}放到"白骨精"后，很明显当前数组的容量已经不够，要扩容了，不执行该句代码了；

addAll(list..哪吒三兄弟, 4);//从第五个位置将“哪吒三兄弟”插进去，那么数组第五个位置后的元素需往后移动三位，数组按规则扩容为 18。

铁扇	牛魔王	蛟魔王	鹏魔王	金吒	木吒	哪吒	狮驼王	猕猴王	禺狨王	美猴王	白骨精						

指定了插入位置的，会通过**rangeCheckForAdd(int index)**方法判断是否数组越界

2. set(int index, E element)

因为 ArrayList 底层是由数组实现的，set 实现非常简单，调用 set(8, "猪八戒") 通过传入的数字下标到对应的位置，替换其中的元素，前提也需要首先判断传入的数组下标是否越界。将“猕猴王”替换为“八戒”。

```
public E set(int index, E element) {
    rangeCheck(index);
    E oldValue = elementData(index);
    elementData[index] = element;
    return oldValue;
}
```

//返回值“猕猴王”，当前数组中数据：

铁扇	牛魔王	蛟魔王	鹏魔王	金吒	木吒	哪吒	狮驼王	猪八戒	禺狨王	美猴王	白骨精						

3. get(int index)

ArrayList 中 get 方法也非常简单，通过下标查找即可，同时需要进行了类型转换，因为数组为 Objec[]，前提是需要判断传入的数组下标是否越界。

```
public E get(int index) {
    rangeCheck(index);
    return elementData(index);
}
E elementData(int index) {
    return (E) elementData[index];
}
```

调用 get(6)返回”哪吒“。

4. remove(int index)

首先说一下 ArrayList 通过下标删除的方法，我们看一下源码

```
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);
    elementData[--size] = null; // clear to let GC do its work
    return oldValue;
}
```

通过源码我们可以看到首先获取了待删除的元素，并最终返回了。其次计算了数组中需要移动的位数 size - index - 1，那么很明显我们可以得出待删除的是最后一个元素的话，移到位数为 0，否则移动位大于 0，那么通过数组元素的拷贝来实现往前移动相应位数。

如 remove(10)，找到的元素为“美猴王”，那么移动位数 = 12-10-1 = 1；此时将原本在第 12 个置上（数组下标为 11）的“白骨精”往前移动一位，同时设置 elementData[11] = null；这里通过置 null 值让 GC 起作用。

5. remove(Object o)

删除 ArrayList 中的值对象，其实和通过下标删除很相似，只是多了一个步骤，遍历底层数组 elementData，通过 equals()方法或 == （特殊情况下）来找到要删除的元素，获取其下标，调用 remove(int index)一样的代码即可。

```
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;
}
```

6. 其他方法

size() : 获取集合长度，通过定义在 ArrayList 中的私有变量 size 得到

isEmpty()：是否为空，通过定义在 ArrayList 中的私有变量 size 得到

contains(Object o)：是否包含某个元素，通过遍历底层数组 elementData，通过 equals 或 == 进判断

clear()：集合清空，通过遍历底层数组 elementData，设置为 null

三。 总结

本文主要讲解了 ArrayList 原理，从底层数组着手，讲解了 ArrayList 定义时到底发生了什么，再添元素时，扩容规则如何，删除元素时，数组的元素的移动方式以及一些常用方法的用途，若有不对之

，请批评指正，望共同进步，谢谢！