

第二章 ArrayList源码解析

一、对于ArrayList需要掌握的七点内容

- ArrayList的创建：即构造器
- 往ArrayList中添加对象：即add(E)方法
- 获取ArrayList中的单个对象：即get(int index)方法
- 删除ArrayList中的对象：即remove(E)方法
- 遍历ArrayList中的对象：即iterator，在实际中更常用的是增强型的for循环去做遍历
- 判断对象是否存在于ArrayList中：contain(E)
- ArrayList中对象的排序：主要取决于所采取的排序算法（以后讲）

二、源码分析

2.1、ArrayList的创建（常见的两种方式）

```
List<String> strList = new ArrayList<String>();
List<String> strList2 = new ArrayList<String>(2);
```



ArrayList源代码：

基本属性：

```
//对象数组：ArrayList的底层数据结构
private transient Object[] elementData;
//elementData中已存放的元素的个数，注意：不是elementData的容量
private int size;
```

注意：

- **transient关键字的作用**：在采用**Java默认**的序列化机制的时候，被该关键字修饰的属性不会被序列化。
- ArrayList类实现了java.io.Serializable接口，即采用了Java默认的序列化机制
- 上面的elementData属性采用了transient来修饰，表明其不使用Java默认的序列化机制来实例化，但是该属性是ArrayList的底层数据结构，在网络传输中一定需要将其序列化，之后使用的时候还需要反序列化，那不采用Java默认的序列化机制，那采用什么呢？直接翻到源码的最下边有两个方法，发现ArrayList自己实现了序列化和反序列化的方法



```
/**
 * Save the state of the <tt>ArrayList</tt> instance to a stream (that is,
 * serialize it).
 *
 * @serialData The length of the array backing the <tt>ArrayList</tt>
 *               instance is emitted (int), followed by all of its elements
 *               (each an <tt>Object</tt>) in the proper order.
 */
private void writeObject(java.io.ObjectOutputStream s)
    throws java.io.IOException {
    // Write out element count, and any hidden stuff
    int expectedModCount = modCount;
    s.defaultWriteObject();

    // Write out array length
    s.writeInt(elementData.length);

    // Write out all elements in the proper order.
    for (int i = 0; i < size; i++)
        s.writeObject(elementData[i]);

    if (modCount != expectedModCount) {
        throw new ConcurrentModificationException();
    }
}

/**
 * Reconstitute the <tt>ArrayList</tt> instance from a stream (that is,
 * deserialize it).
 */
```

```
private void readObject(java.io.ObjectInputStream s)
    throws java.io.IOException, ClassNotFoundException {
    // Read in size, and any hidden stuff
    s.defaultReadObject();

    // Read in array length and allocate array
    int arrayLength = s.readInt();
    Object[] a = elementData = new Object[arrayLength];

    // Read in all elements in the proper order.
    for (int i = 0; i < size; i++)
        a[i] = s.readObject();
}
```

构造器：

```
/**
 * 创建一个容量为initialCapacity的空 ( size==0 ) 对象数组
 */
public ArrayList(int initialCapacity) {
    super(); //即父类protected AbstractList() {}
    if (initialCapacity < 0)
        throw new IllegalArgumentException("Illegal Capacity:" + initialCapacity);
    this.elementData = new Object[initialCapacity];
}

/**
 * 默认初始化一个容量为10的对象数组
 */
public ArrayList() {
    this(10); //即上边的public ArrayList(int initialCapacity){}构造器
}
```

在我们执行new ArrayList<String>()时，会调用上边的无参构造器，创建一个容量为10的对象数组。

在我们执行new ArrayList<String>(2)时，会调用上边的public ArrayList(int initialCapacity)，创建一个容量为2的对象数组。

注意：

- 上边有参构造器的super()方法是ArrayList父类AbstractList的构造方法，这个构造方法如下，是一个空构造方法：

```
protected AbstractList() {
}
```

- 在实际使用中，如果我们能对所需的ArrayList的大小进行判断，有两个好处：
 - 节省内存空间（eg.我们只需要放置两个元素到数组，new ArrayList<String>(2)）
 - 避免数组扩容（下边会讲）引起的效率下降（eg.我们只需要放置大约37个元素到数组，new ArrayList<String>(40)）

2.2、往ArrayList中添加对象（常见的两个方法add(E)和addAll(Collection<? extends E> c)）

2.2.1、add(E)

```
strList2.add("hello");
```

ArrayList源代码：

```
/**
 * 向elementData中添加元素
 */
public boolean add(E e) {
    ensureCapacity(size + 1); //确保对象数组elementData有足够的容量，可以将新加入的元素e加进去
    elementData[size++] = e; //加入新元素e，size加1
    return true;
}
```

```
    }
}

/**
 * 确保数组的容量足够存放新加入的元素，若不够，要扩容
 */
public void ensureCapacity(int minCapacity) {
    modCount++;
    int oldCapacity = elementData.length; //获取数组大小（即数组的容量）
    //当数组满了，又有新元素加入的时候，执行扩容逻辑
    if (minCapacity > oldCapacity) {
        Object oldData[] = elementData;
        int newCapacity = (oldCapacity * 3) / 2 + 1; //新容量为旧容量的1.5倍+1
        if (newCapacity < minCapacity) //如果扩容后的新容量还是没有传入的所需的最小容量大或等于（主要发生在addAll(Collection<? extends E> c)中）
            newCapacity = minCapacity; //新容量设为最小容量
        elementData = Arrays.copyOf(elementData, newCapacity); //复制新容量
    }
}
```

在上述代码的扩容结束后，调用了Arrays.copyOfOf(elementData, newCapacity)方法，这个方法中：对于我们这里而言，先创建了一个新的容量为newCapacity的对象数组，然后使用System.arraycopy()方法将旧的对象数组复制到新的对象数组中去了。

注意：

- modCount变量用于在遍历集合（iterator()）时，检测是否发生了add、remove操作。

2.2.2、addAll(Collection<? extends E> c)

使用方式：

```
List<String> strList = new ArrayList<String>();
strList.add("jigang");
strList.add("nana");
strList.add("nana2");

List<String> strList2 = new ArrayList<String>(2);
strList2.addAll(strList);
```

源代码：

```
/**
 * 将c全部加入elementData
 */
public boolean addAll(Collection<? extends E> c) {
    Object[] a = c.toArray(); //将c集合转化为对象数组a
    int numNew = a.length; //获取a对象数组的容量
    ensureCapacity(size + numNew); //确保对象数组elementData有足够的容量，可以将新加入的a对象数组加进去
    System.arraycopy(a, 0, elementData, size, numNew); //将对象数组a拷贝到elementData中去
    size += numNew; //重新设置elementData中已加入的元素的个数
    return numNew != 0; //若加入的是空集合则返回false
}
```

注意：

- 从上述代码可以看出，若加入的c是空集合，则返回false
- ensureCapacity(size + numNew);这个方法在上边讲
- System.arraycopy()方法定义如下：

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

将数组src从下标为srcPos开始拷贝，一直拷贝length个元素到dest数组中，在dest数组中从destPos开始加入先的srcPos数组元素。

除了以上两种常用的add方法外，还有如下两种：

2.2.3、 add(int index, E element)




```
/**
 * 在特定位置（只能是已有元素的数组的特定位置）index插入元素E
 */
public void add(int index, E element) {
    //检查index是否在已有的数组中
    if (index > size || index < 0)
        throw new IndexOutOfBoundsException("Index:"+index+",Size:"+size);
    ensureCapacity(size + 1); //确保对象数组elementData有足够的容量，可以将新加入的元素e加进去
    System.arraycopy(elementData, index, elementData, index+1, size-index); //将index及其后边的所有的元素整块
    //后移，空出index位置
    elementData[index] = element; //插入元素
    size++; //已有数组元素个数+1
}
```




注意：

- **index<=size才行，并不是index<elementData.length**

2.2.4、 set(int index, E element)



```
/**
 * 更换特定位置index上的元素为element，返回该位置上的旧值
 */
public E set(int index, E element) {
    RangeCheck(index); //检查索引范围
    E oldValue = (E) elementData[index]; //旧值
    elementData[index] = element; //该位置替换为新值
    return oldValue; //返回旧值
}
```




2.3、 获取ArrayList中的单个对象（ get(int index)）


实现方式：

```
ArrayList<String> strList2 = new ArrayList<String>(2);
strList2.add("hello");
strList2.add("nana");
strList2.add("nana2");
System.out.println(strList2.get(0));
```

源代码：



```
/**
 * 按照索引查询对象E
 */
public E get(int index) {
    RangeCheck(index); //检查索引范围
    return (E) elementData[index]; //返回元素，并将Object转型为E
}
```





```
/**
 * 检查索引index是否超出size-1
 */
private void RangeCheck(int index) {
    if (index >= size)
        throw new IndexOutOfBoundsException("Index:"+index+",Size:"+size);
}
```



注：这里对index进行了索引检查，是为了将异常内容写的详细一些并且将检查的内容缩小（index<0||index>=size，注意这里的size是已存储元素的个数）；

事实上不检查也可以，因为对于数组而言，如果index不满足要求（index<0||index>=length，注意这里的length是数组的容量），都会直接抛出数组越界异常，而假设数组的length为10，当前的size是2，你去计算array[9]，这时候得出是null，这也是上边get为什么减小检查范围的原因。

2.4、删除ArrayList中的对象

2.4.1、remove(Object o)

使用方式：

```
strList2.remove("hello");
```

源代码：



```
/**
 * 从前向后移除第一个出现的元素o
 */
public boolean remove(Object o) {
    if (o == null) { //移除对象数组elementData中的第一个null
        for (int index = 0; index < size; index++)
            if (elementData[index] == null) {
                fastRemove(index);
                return true;
            }
    } else { //移除对象数组elementData中的第一个o
        for (int index = 0; index < size; index++)
            if (o.equals(elementData[index])) {
                fastRemove(index);
                return true;
            }
    }
    return false;
}

/**
 * 删除单个位置的元素，是ArrayList的私有方法
 */
private void fastRemove(int index) {
    modCount++;
    int numMoved = size - index - 1;
    if (numMoved > 0) //删除的不是最后一个元素
        System.arraycopy(elementData, index + 1, elementData, index, numMoved); //删除的元素到最后的元素整块前移

    elementData[--size] = null; //将最后一个元素设为null，在下次gc的时候就会回收掉了
}
```



2.4.2、remove(int index)

使用方式：

```
strList2.remove(0);
```

源代码：



```
/**
 * 删除指定索引index下的元素，返回被删除的元素
 */
public E remove(int index) {
    RangeCheck(index); //检查索引范围

    E oldValue = (E) elementData[index]; //被删除的元素
    fastRemove(index);
    return oldValue;
}
```




注意：

- remove(Object o)需要遍历数组，remove(int index)不需要，只需要判断索引符合范围即可，所以，通常：后者效率更高。

2.5、判断对象是否存在于ArrayList中 (contains(E))

源代码：



```
/**
 * 判断动态数组是否包含元素o
 */
public boolean contains(Object o) {
    return indexOf(o) >= 0;
}

/**
 * 返回第一个出现的元素o的索引位置
 */
public int indexOf(Object o) {
    if (o == null) { //返回第一个null的索引
        for (int i = 0; i < size; i++)
            if (elementData[i] == null)
                return i;
    } else { //返回第一个o的索引
        for (int i = 0; i < size; i++)
            if (o.equals(elementData[i]))
                return i;
    }
    return -1; //若不包含，返回-1
}

/**
 * 返回最后一个出现的元素o的索引位置
 */
public int lastIndexOf(Object o) {
    if (o == null) {
        for (int i = size - 1; i >= 0; i--)
            if (elementData[i] == null)
                return i;
    } else {
        for (int i = size - 1; i >= 0; i--)
            if (o.equals(elementData[i]))
                return i;
    }
}
```

```
        return -1;
    }
}
```



注意：

- indexOf(Object o)返回第一个出现的元素o的索引；lastIndexOf(Object o)返回最后一个o的索引

2.6、遍历ArrayList中的对象 (iterator())

使用方式：



```
List<String> strList = new ArrayList<String>();
strList.add("jigang");
strList.add("nana");
strList.add("nana2");

Iterator<String> it = strList.iterator();
while (it.hasNext()) {
    System.out.println(it.next());
}
```



源代码：iterator()方法是在AbstractList中实现的，该方法返回AbstractList的一个内部类Itr对象

```
public Iterator<E> iterator() {
    return new Itr(); //返回一个内部类对象
}
```

Itr：



```
private class Itr implements Iterator<E> {

    int cursor = 0; //标记位：标记遍历到哪一个元素
    int expectedModCount = modCount; //标记位：用于判断是否在遍历的过程中，是否发生了add、remove操作

    //检测对象数组是否还有元素
    public boolean hasNext() {
        return cursor != size(); //如果cursor==size，说明已经遍历完了，上一次遍历的是最后一个元素
    }

    //获取元素
    public E next() {
        checkForComodification(); //检测在遍历的过程中，是否发生了add、remove操作
        try {
            E next = get(cursor++);
            return next;
        } catch (IndexOutOfBoundsException e) { //捕获get(cursor++)方法的IndexOutOfBoundsException
            checkForComodification();
            throw new NoSuchElementException();
        }
    }

    //检测在遍历的过程中，是否发生了add、remove等操作
    final void checkForComodification() {
        if (modCount != expectedModCount) //发生了add、remove操作,这个我们可以查看add等的源代码，发现会出现
modCount++

            throw new ConcurrentModificationException();
        }
    }
}
```



遍历的整个流程结合"使用方式"与"Itr的注释"来看。注：上述的Itr我去掉了一个此时用不到的方法和属性。

三、总结

- ArrayList基于数组方式实现，无容量的限制（会扩容）
- 添加元素时可能要扩容（所以最好预判一下），删除元素时不会减少容量（若希望减少容量，trimToSize()），**删除元素时，将删除掉的位置元素置为null，下次gc就会回收这些元素所占的内存空间。**
- 线程不安全
- add(int index, E element)：添加元素到数组中指定位置的时候，需要将该位置及其后边所有的元素都整块向后复制一位
- get(int index)：获取指定位置上的元素时，可以通过索引直接获取（O(1)）
- remove(Object o)需要遍历数组
- remove(int index)不需要遍历数组，只需判断index是否符合条件即可，效率比remove(Object o)高
- contains(E)需要遍历数组

做以上总结，主要是为了与后边的LinkedList作比较。

elementData