链滴

ArrayList 源码学习

作者：lvtaos

原文链接：https://ld246.com/article/1564190878978

来源网站：链滴

<h2 id="ArrayList-源码分析">ArrayList 源码分析</h2>
当前分析的源码版本为
<pre><code class="highlight-chroma">jdk 1.7.0_80
</code></pre>
<code>ArrayList</code> 是一个动态数组容器类。下方的代码为 <code>ArrayList</code> 的定义。
<pre><code class="highlight-chroma">public class ArrayList<E> extends AbstractList<E>
        implements List<E>, RandomAccess, Cloneable, java.io.Serializable
{
    // TODO ...
}
</code></pre>
<h2 id="一--方法的扩容原理是什么">一、<code>add()</code> 方法的扩容原理是什么</h2>
先看源码
<pre><code class="highlight-chroma">public boolean add(E e) {
    ensureCapacityInternal(size + 1); // Increments modCount!!
    elementData[size++] = e;
    return true;
}
</code></pre>
代码中的 <code>elementData</code> 和 <code>size</code> 是 <code>ArrayList</code> 类的两个成员变量。
<pre><code class="highlight-chroma">transient Object[] elementData; // non-private to simplify nested class access
private int size;
</code></pre>
代码注释中的 <code>Increments modCount!!</code> 在遍历的方法中再讲。<code>modCount</code> 代表容器被修改的次数。
首先调用 <code>ensureCapacityInternal(size + 1)</code> 方法确保容量是够的。然后对 <code>elementData[size++]</code> 进行赋值，最后 <code>size</code> 加 1。以下为 <code>ensureCapacityInternal()</code> 方法的代码。
<pre><code class="highlight-chroma">private void ensureCapacityInternal(int minCapacity) {
    ensureExplicitCapacity(calculateCapacity(elementData, minCapacity));
}

private static int calculateCapacity(Object[] elementData, int minCapacity) {
    if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
        return Math.max(DEFAULT_CAPACITY, minCapacity);
    }
    return minCapacity;
}

</code></pre>

private void ensur
ExplicitCapacity(int minCapacity) {
    modCount++;

    // overflow-con
cious code
    if (minCapacity
 elementData.length > 0)
        grow(minCa
acity);
}
</code></pre>
<code>grow()</code> 方法的主要内容为
<pre><code class="highlight-chroma"><span class="highlight
cl">private void grow(int minCapacity) {
    // overflow-con
cious 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 i
 usually close to size, so this is a win:
    elementData =
Arrays.copyOf(elementData, newCapacity);
}
</code></pre>
扩容的核心的代码是 <code>int newCapacity = oldCapacity + (oldCapacity >> 1);</
ode>，右移 <code>>></code> 一位相当于除以 2。可以看出容器的容量扩大了原来的 1.5
倍。
<h2 id="二--方法">二、<code>remove()</code> 方法</h2>
<pre><code class="highlight-chroma"><span class="highlight
cl">public E remove(int index) {
    rangeCheck(ind
x);

    modCount++;
    E oldValue = el
mentData(index);

    int numMoved
 size - index - 1;
    if (numMoved
gt; 0)
        System.array
opy(elementData, index+1, elementData, index,

```
</span></span><span class="highlight-line"><span class="highlight-cl">                    n
mMoved);
</span></span><span class="highlight-line"><span class="highlight-cl">    elementData[--s
ze] = null; // clear to let GC do its work
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    return oldValue;
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">private void rang
Check(int index) {
</span></span><span class="highlight-line"><span class="highlight-cl">    if (index &gt;= s
ze)
</span></span><span class="highlight-line"><span class="highlight-cl">        throw new In
exOutOfBoundsException(outOfBoundsMsg(index));
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p><code>remove()</code> 方法的代码相对简单，主要是调用了 <code>System.arraycopy()<
code> 方法对数组 <code>elementData</code> 自身进行复制，起始位置从 <code>index+1</
ode> 复制到 <code>index</code> 。然后将索引位置在 <code>size--</code> 的元素的引用
为 <code>null</code>， 最后 <code>size</code> 减一。</p>
<h2 id="三--中的向前遍历">三、<code>ArraysList</code> 中的向前遍历</h2>
<p><code>ArrayList</code> 类中维护了一个私有的内部类 <code>ListItr</code>，<code>Lis
Itr</code> 实现了 <code>ListIterator</code> 接口。以下为 <code>ListItr</code> 的定义。<
p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">private class ListItr extends Itr implements ListIterator&lt;E&gt; {
</span></span><span class="highlight-line"><span class="highlight-cl">    // TODO ...
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>接口 <code>ListIterator</code> 继承了 <code>Iterator</code> 接口，并新增了几个新的
法。下面是 <code>ListIterator</code> 的方法。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">public interface ListIterator&lt;E&gt; extends Iterator&lt;E&gt; {
</span></span><span class="highlight-line"><span class="highlight-cl">    boolean hasNex
();
</span></span><span class="highlight-line"><span class="highlight-cl">    E next();
</span></span><span class="highlight-line"><span class="highlight-cl">    boolean hasPre
ious();
</span></span><span class="highlight-line"><span class="highlight-cl">    E previous();
</span></span><span class="highlight-line"><span class="highlight-cl">    int nextIndex();
</span></span><span class="highlight-line"><span class="highlight-cl">    int previousInd
x();
</span></span><span class="highlight-line"><span class="highlight-cl">    void remove();
</span></span><span class="highlight-line"><span class="highlight-cl">    void set(E e);
</span></span><span class="highlight-line"><span class="highlight-cl">    void add(E e);
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>来看一个简单的降序遍历的例子。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">public static void main(String[] args) {
</span></span><span class="highlight-line"><span class="highlight-cl">    List&lt;Integer
gt; list = new ArrayList&lt;&gt;(Arrays.asList(1, 2, 3));
</span></span><span class="highlight-line"><span class="highlight-cl">    ListIterator&lt;I
teger&gt; descItr = list.listIterator(list.size());
```

    while (descItr.hasPrevious()) {
        Integer previous = descItr.previous();
        System.out.println(previous);
    }
}
</code></pre>
输出
<pre><code class="highlight-chroma">3
2
1
</code></pre>
<h2 id="四--有什么作用">四、<code>modCount</code> 有什么作用</h2>
<code>modCount</code> 是 <code>AbstractList</code> 中的一个成员变量，记录容器结构发生变化的次数，使容器结构发生变化的方法比如有 <code>add()</code> 和 <code>remove()</code> 方法。
<pre><code class="highlight-chroma">protected transient int modCount = 0;
</code></pre>
接下要从方法 <code>iterator()</code> 说起。
<pre><code class="highlight-chroma">public Iterator<E> iterator() {
    return new Itr();
}
</code></pre>
<code>Itr</code> 是 <code>ArrayList</code> 中的一个内部类，实现了 <code>Iterator</code> 接口。
<pre><code class="highlight-chroma">private class Itr implements Iterator<E> {
    int cursor;   // index of next element to return
    int lastRet = -1; // index of last element returned; -1 if no such
    int expectedModCount = modCount;

    public boolean hasNext(){...}
    public E next() {...}
    public void remove() {...}
    final void checkForComodification() {...}
}
</code></pre>
<code>cursor</code> 是下一个要返回的元素的索引值。<code>lastRet</code> 是最后一要返回的元素的索引值。<code>expectModCount</code> 表示期望地修改次数。在初始化迭代的时候复赋值为 <code>modCount</code>。
下面代码中要通过迭代器删除 <code>list</code> 中的一个元素，并解释迭代器中每个方法的作原理。

<pre><code class="highlight-chroma"><span class="highlight
cl">public static void main(String[] args) {
    List<Integer
gt; list = new ArrayList<Integer>();
    list.add(1);
    list.add(2);
    list.add(3);
    Iterator<Inte
er> iterator = list.iterator();
    Integer next;
    while(iterator.h
sNext()) {
        next = iterato
.next();
        if (next == 3)
{
            iterator.re
ove();
        }
    }
    System.out.print
n(list.size());
}
</code></pre>
<code>while</code> 循环中，先执行 <code>hasNext()</code> 方法，判断元素的索引位
是否超过容器容量。
<pre><code class="highlight-chroma"><span class="highlight
cl">public boolean hasNext() {
    return cursor !=
size;
}
</code></pre>
<pre><code class="highlight-chroma"><span class="highlight
cl">public E next() {
    checkForComod
fication();
    int i = cursor;
    if (i >= size)
        throw new N
SuchElementException();
    Object[] elemen
Data = ArrayList.this.elementData;
    if (i >= elem
ntData.length)
        throw new C
ncurrentModificationException();
    cursor = i + 1;
    return (E) elem
ntData[lastRet = i];
}

final void checkFo
Comodification() {
    if (modCount !=

expectedModCount)
        throw new C
ncurrentModificationException();
}
</code></pre>
<code>next()</code> 方法先调用 <code>checkForComodification()</code> 检查容器的构是否发生变化。在方法 <code>checkForComodification()</code> 中，如果 <code>modCoun != expectedModCount</code> 就抛出异常 <code>throw new ConcurrentModificationExcepton(); </code>。检查成功后对下一个元素的索引变量 <code>cursor</code> 进行赋值，然后赋值变量 <code>lastRet</code>，最后返回当前索引位置所在的元素。
下面是删除方法 <code>remove()</code> 的源代码。
<pre><code class="highlight-chroma"><span class="highlight
cl">public void remove() {
    if (lastRet < 0

        throw new Ill
galStateException();
    checkForComod
fication();

    try {
        ArrayList.this.
emove(lastRet);
        cursor = last
et;
        lastRet = -1;
        expectedMo
Count = modCount;
    } catch (IndexO
tOfBoundsException ex) {
        throw new C
ncurrentModificationException();
    }
}
</code></pre>
执行 <code>remove()</code> 方法的第一步就是对 <code>lastRet</code> 的检查。所以在使用迭代器遍历时，调用 <code>remove()</code> 方法前一定要调用 <code>next()</ode> 方法。在 <code>next()</code> 方法的结尾处会对 <code>lastRet</code> 方进行赋值，否则 <code>lastRet</code> 的初始值为 -1，就会抛出异常 <code>java.lang.IllegalSteException</code>。
接下来调用 <code>ArrayList</code> 的 <code>remove()</code> 方法。然后对 <code>epectModCount</code> 进行赋值，保证容器结构的一致。
<h2 id="五-为什么在--方法中删除元素会抛异常">五、为什么在 <code>foreach</code> 方法中除元素会抛异常</h2>
下面使用 <code>foreach</code> 语法删除一个元素。
<pre><code class="highlight-chroma"><span class="highlight
cl">public static void main(String[] args) {
    List<Integer
gt; list = new ArrayList<>(5);
    list.add(1);
    list.add(2);
    list.add(3);
    list.add(4);
    list.add(5);

```
</span></span><span class="highlight-line"><span class="highlight-cl">    for (Integer i : li
t) {
</span></span><span class="highlight-line"><span class="highlight-cl">        if (i == 5) {
</span></span><span class="highlight-line"><span class="highlight-cl">            list.remove
i);
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>运行后控制台抛出异常。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">Exception in thread "main" java.util.ConcurrentModificationException
</span></span></code></pre>
<p><code>foreach</code> 是 java 中的一颗语法糖，编译后的字节码文件是这样的。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">public static void main(String[] args) {
</span></span><span class="highlight-line"><span class="highlight-cl">    List&amp;lt;Integer
gt; list = new ArrayList();
</span></span><span class="highlight-line"><span class="highlight-cl">    list.add(1);
</span></span><span class="highlight-line"><span class="highlight-cl">    list.add(2);
</span></span><span class="highlight-line"><span class="highlight-cl">    list.add(3);
</span></span><span class="highlight-line"><span class="highlight-cl">    Iterator&amp;lt;Inte
er&amp;gt; iterator = list.iterator();
</span></span><span class="highlight-line"><span class="highlight-cl">    System.out.print
n(list.size());
</span></span><span class="highlight-line"><span class="highlight-cl">    List&amp;lt;Integer
gt; list1 = new ArrayList();
</span></span><span class="highlight-line"><span class="highlight-cl">    list1.add(1);
</span></span><span class="highlight-line"><span class="highlight-cl">    list1.add(2);
</span></span><span class="highlight-line"><span class="highlight-cl">    list1.add(3);
</span></span><span class="highlight-line"><span class="highlight-cl">    Iterator i$ = list
.iterator();
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    while(i$.hasNex
()) {
</span></span><span class="highlight-line"><span class="highlight-cl">        Integer integ
r = (Integer)i$.next();
</span></span><span class="highlight-line"><span class="highlight-cl">        if (integer ==
3) {
</span></span><span class="highlight-line"><span class="highlight-cl">            list1.remo
e(integer);
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>上面代码中的 <code>remove()</code> 方法如下。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">public boolean remove(Object o) {
</span></span><span class="highlight-line"><span class="highlight-cl">    if (o == null) {
</span></span><span class="highlight-line"><span class="highlight-cl">        for (int index
= 0; index &amp;lt; size; index++)
</span></span><span class="highlight-line"><span class="highlight-cl">            if (elemen
Data[index] == null) {
</span></span><span class="highlight-line"><span class="highlight-cl">                fastRem
```

```
ve(index);
</span></span><span class="highlight-line"><span class="highlight-cl">                return t
ue;
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">    } else {
</span></span><span class="highlight-line"><span class="highlight-cl">        for (int index
= 0; index &amp;lt; size; index++)
</span></span><span class="highlight-line"><span class="highlight-cl">            if (o.equals
elementData[index])) {
</span></span><span class="highlight-line"><span class="highlight-cl">                fastRem
ve(index);
</span></span><span class="highlight-line"><span class="highlight-cl">                return t
ue;
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    return false;
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">private void fastR
move(int index) {
</span></span><span class="highlight-line"><span class="highlight-cl">    modCount++;
</span></span><span class="highlight-line"><span class="highlight-cl">    int numMoved
 size - index - 1;
</span></span><span class="highlight-line"><span class="highlight-cl">    if (numMoved
gt; 0)
</span></span><span class="highlight-line"><span class="highlight-cl">        System.array
opy(elementData, index+1, elementData, index,
</span></span><span class="highlight-line"><span class="highlight-cl">                n
mMoved);
</span></span><span class="highlight-line"><span class="highlight-cl">    elementData[--s
ze] = null; // clear to let GC do its work
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>程序最后走到了 <code>fastRemove()</code> 方法中。<code>fastRemove()</code> 方
和迭代器中的 <code>remove()</code> 方法不同，程序中并没有维护 <code>expectedModCou
t = modCount</code>。所以在 <code>next()</code> 方法中执行 <code>checkForComodific
tion()</code>，发现 <code>expectedModCount</code> 和 <code>modCount</code> 不
，容器的结构发生了变化，就会抛出异常 <code>throw new ConcurrentModificationException()
/code>。</p>
<h2 id="六--中的其他方法">六、<code>ArrayList</code> 中的其他方法</h2>
<p>在开发过程中还有一些 <code>ArraysList</code> 中的常用方法。</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">// 构造器方法
</span></span><span class="highlight-line"><span class="highlight-cl">public ArrayList(int
initialCapacity) {...}
</span></span><span class="highlight-line"><span class="highlight-cl">public ArrayList() {.
.}
</span></span><span class="highlight-line"><span class="highlight-cl">public ArrayList(C
llection&amp;lt;? extends E&amp;gt; c) {...}
</span></span><span class="highlight-line"><span class="highlight-cl">// 判断容器中元素
个数
</span></span><span class="highlight-line"><span class="highlight-cl">public int size() {...}
</span></span><span class="highlight-line"><span class="highlight-cl">// 判断元素是否为
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">public boolean is
mpty() {...}
</span></span><span class="highlight-line"><span class="highlight-cl">// 是否包含某个元

</span></span><span class="highlight-line"><span class="highlight-cl">public boolean co
tains(Object o) {...}
</span></span><span class="highlight-line"><span class="highlight-cl">// 找出元素的索引
置
</span></span><span class="highlight-line"><span class="highlight-cl">public int indexOf
Object o) {...}
</span></span><span class="highlight-line"><span class="highlight-cl">public int lastInde
Of(Object o) {...}
</span></span><span class="highlight-line"><span class="highlight-cl">// 将容易转换为数

</span></span><span class="highlight-line"><span class="highlight-cl">public Object[] to
rray() {...}
</span></span><span class="highlight-line"><span class="highlight-cl">public &amp;lt;T&amp;gt; T[
 toArray(T[] a) {...}
</span></span><span class="highlight-line"><span class="highlight-cl">// 获取元素
</span></span><span class="highlight-line"><span class="highlight-cl">public E get(int in
ex) {...}
</span></span><span class="highlight-line"><span class="highlight-cl">// 替换元素
</span></span><span class="highlight-line"><span class="highlight-cl">public E set(int in
ex, E element) {...}
</span></span></code></pre>
<p>（完）</p>
```