

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路 [最近阅读](#)

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

## 05 ArrayList 源码解析和设计思路

更新时间：2019-11-26 09:44:51



“

耐心和恒心总会得到报酬的。

——爱因斯坦

”

### 引导语

ArrayList 我们几乎每天都会使用到，但真正面试的时候，发现还是有不少人对源码细节说不清楚，给面试官留下比较差的印象，本小节就和大家一起看看面试中和 ArrayList 相关的源码。

### 1 整体架构

ArrayList 整体架构比较简单，就是一个数组结构，比较简单，如下图：

index	0	1	2	3	4	5	6	7	8	9
elementData	O1	O2	O3							

图中展示是长度为 10 的数组，从 1 开始计数，index 表示数组的下标，从 0 开始计数，elementData 表示数组本身，源码中除了这两个概念，还有以下三个基本概念：

- DEFAULT\_CAPACITY 表示数组的初始大小，默认是 10，这个数字要记住；
- size 表示当前数组的大小，类型 int，没有使用 volatile 修饰，非线程安全的；
- modCount 统计当前数组被修改的版本次数，数组结构有变动，就会 +1。

#### 类注释

看源码，首先要看类注释，我们看看类注释上面都说了什么，如下：

- 允许 put null 值，会自动扩容；
- size、isEmpty、get、set、add 等方法时间复杂度都是 O(1)；

<div><div>← 慕课专栏</div><div>三 面试官系统精讲Java源码及大厂真题 / 05 ArrayList 源码解析和设计思路</div></div>	
目录	
第1章 基础	
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	<div><ul style="list-style-type: none"><li>• 增强 for 循环，或者使用迭代器迭代过程中，如果数组大小被改变，会快速失败，抛出异常。</li></ul></div> <p>除了上述注释中提到的 4 点，初始化、扩容的本质、迭代器等问题也经常被问，接下来我们从源码出发，一一解析。</p>
第2章 集合	<h2>2 源码解析</h2> <h3>2.1 初始化</h3> <p>我们有三种初始化办法：无参数直接初始化、指定大小初始化、指定初始数据初始化，源码如下：</p> <pre>private static final Object[] DEFAULTCAPACITY_EMPTY_ELEMENTDATA = {};  //无参数直接初始化，数组大小为空 public ArrayList() {     this.elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA; }  //指定初始数据初始化 public ArrayList(Collection&lt;? extends E&gt; c) {     //elementData 是保存数组的容器，默认为 null     elementData = c.toArray();     //如果给定的集合（c）数据有值     if ((size = elementData.length) != 0) {         // c.toArray might (incorrectly) not return Object[] (see 6260652)         //如果集合元素类型不是 Object 类型，我们会转成 Object         if (elementData.getClass() != Object[].class) {             elementData = Arrays.copyOf(elementData, size, Object[].class);         }     } else {         // 给定集合（c）无值，则默认空数组         this.elementData = EMPTY_ELEMENTDATA;     } }</pre>
05 ArrayList 源码解析和设计思路 <a href="#">最近阅读</a>	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	<p>除了源码的中文注释，我们补充两点：</p> <p>1：ArrayList 无参构造器初始化时，默认大小是空数组，并不是大家常说的 10，10 是在第一次 add 的时候扩容的数组值。</p> <p>2：指定初始数据初始化时，我们发现一个样子的注释 see 6260652，这是 Java 的一个 bug，意思是当给定集合内的元素不是 Object 类型时，我们会转化成 Object 的类型。一般情况下都不会触发此 bug，只有在下列场景下才会触发：ArrayList 初始化之后（ArrayList 元素非 Object 类型），再次调用 toArray 方法，得到 Object 数组，并且往 Object 数组赋值时，才会触发此 bug，代码和原因如图：</p>
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路 [最近阅读](#)

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用



官方查看文档地址：[https://bugs.java.com/bugdatabase/view\\_bug.do?bug\\_id=6260652](https://bugs.java.com/bugdatabase/view_bug.do?bug_id=6260652)，问题在 Java 9 中被解决。

2.2 新增和扩容实现

新增就是往数组中添加元素，主要分成两步：

- 判断是否需要扩容，如果需要执行扩容操作；
- 直接赋值。

两步源码体现如下：

```
public boolean add(E e) {
    //确保数组大小是否足够，不够执行扩容，size 为当前数组的大小
    ensureCapacityInternal(size + 1); // Increments modCount!!
    //直接赋值，线程不安全的
    elementData[size++] = e;
    return true;
}
```

我们先看下扩容 ( ensureCapacityInternal ) 的源码：

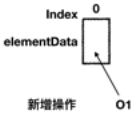
```
private void ensureCapacityInternal(int minCapacity) {
    //如果初始化数组大小时，有给定初始值，以给定的大小为准，不走 if 逻辑
    if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
        minCapacity = Math.max(DEFAULT_CAPACITY, minCapacity);
    }
    //确保容积足够
    ensureExplicitCapacity(minCapacity);
}

private void ensureExplicitCapacity(int minCapacity) {
    //记录数组被修改
    modCount++;
    // 如果我们期望的最小容量大于目前数组的长度，那么就扩容
    if (minCapacity - elementData.length > 0)
        grow(minCapacity);
}

//扩容，并把现有数据拷贝到新的数组里面去
private void grow(int minCapacity) {
    int oldCapacity = elementData.length;
    // oldCapacity >> 1 是把 oldCapacity 除以 2 的意思
    int newCapacity = oldCapacity + (oldCapacity >> 1);

    // 如果扩容后的值 < 我们的期望值，扩容后的值就等于我们的期望值
    if (newCapacity - minCapacity < 0)
        newCapacity = minCapacity;

    // 如果扩容后的值 > jvm 所能分配的数组的最大值，那么就用 Integer 的最大值
    if (newCapacity - MAX_ARRAY_SIZE > 0)
```

<div>← 慕课专栏</div>	<div>面试官系统精讲Java源码及大厂真题 / 05 ArrayList 源码解析和设计思路</div>
<div>目录</div>	<div>elementData = Arrays.<b>copyOf</b>(elementData, newCapacity); }</div>
<div>第1章 基础</div>	<div>注解应该比较详细，我们需要注意的四点是：</div>
<div>01 开篇词：为什么学习本专栏</div>	<div></div>
<div>02 String、Long 源码解析和面试题</div>	<div><ul style="list-style-type: none"><li>扩容的规则并不是翻倍，是原来容量大小 + 容量大小的一半，直白来说，扩容后的大小是原来容量的 1.5 倍；</li></ul></div>
<div>03 Java 常用关键字理解</div>	<div><ul style="list-style-type: none"><li>ArrayList 中的数组的最大值是 Integer.MAX_VALUE，超过这个值，JVM 就不会给数组分配内存空间了。</li></ul></div>
<div>04 Arrays、Collections、Objects 常用方法源码解析</div>	<div><ul style="list-style-type: none"><li>新增时，并没有对值进行严格的校验，所以 ArrayList 是允许 null 值的。</li></ul></div>
<div>第2章 集合</div>	<div>从新增和扩容源码中，下面这点值得我们借鉴：</div>
<div>05 ArrayList 源码解析和设计思路 <a href="#">最近阅读</a></div>	<div><ul style="list-style-type: none"><li>源码在扩容的时候，有数组大小溢出意识，就是说扩容后数组的大小下界不能小于 0，上界不能大于 Integer 的最大值，这种意识我们可以学习。</li></ul></div>
<div>06 LinkedList 源码解析</div>	<div>扩容完成之后，赋值是非常简单的，直接往数组上添加元素即可：elementData [size++] = e。也正是通过这种简单赋值，没有任何锁控制，所以这里的操作是线程不安全的，对于新增和扩容的实现，画了一个动图，如下：</div>
<div>07 List 源码会问哪些面试题</div>	<div></div>
<div>08 HashMap 源码解析</div>	<div></div>
<div>09 TreeMap 和 LinkedHashMap 核心源码解析</div>	<div></div>
<div>10 Map源码会问哪些面试题</div>	<div></div>
<div>11 HashSet、TreeSet 源码解析</div>	<div></div>
<div>12 彰显细节：看集合源码对我们实际工作的帮助和应用</div>	<div></div>
<div>13 差异对比：集合在 Java 7 和 8 有何不同和改进</div>	<div></div>
<div>14 简化工作：Guava Lists Maps 实际工作运用和源码</div>	<div></div>
<div>第3章 并发集合类</div>	<div></div>
<div>15 CopyOnWriteArrayList 源码解析和设计思路</div>	<div></div>
<div>16 ConcurrentHashMap 源码解析和设计思路</div>	<div></div>
<div>17 并发 List、Map源码面试题</div>	<div></div>
<div>18 场景集合：并发 List、Map的应用</div>	<div>System.<b>arraycopy</b>(elementData, 0, newElementData, 0,Math.min(elementData.length,newCap</div>

目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路 [最近阅读](#)

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

ArrayList 删除元素有很多种方式，比如根据数组索引删除、根据值删除或批量删除等等，原理和思路都差不多，我们选取根据值删除方式来进行源码说明：

```
public boolean remove(Object o) {
    // 如果要删除的值是 null，找到第一个值是 null 的删除
    if (o == null) {
        for (int index = 0; index < size; index++)
            if (elementData[index] == null) {
                fastRemove(index);
                return true;
            }
    } else {
        // 如果要删除的值不为 null，找到第一个和要删除的值相等的删除
        for (int index = 0; index < size; index++)
            // 这里是根据 equals 来判断值相等的，相等后再根据索引位置进行删除
            if (o.equals(elementData[index])) {
                fastRemove(index);
                return true;
            }
    }
    return false;
}
```

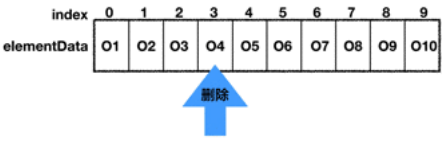
我们需要注意的两点是：

- 新增的时候是没有对 null 进行校验的，所以删除的时候也是允许删除 null 值的；
- 找到值在数组中的索引位置，是通过 equals 来判断的，如果数组元素不是基本类型，需要我们关注 equals 的具体实现。

上面代码已经找到要删除元素的索引位置了，下面代码是根据索引位置进行元素的删除：

```
private void fastRemove(int index) {
    // 记录数组的结构要发生变动了
    modCount++;
    // numMoved 表示删除 index 位置的元素后，需要从 index 后移动多少个元素到前面去
    // 减 1 的原因，是因为 size 从 1 开始算起，index 从 0 开始算起
    int numMoved = size - index - 1;
    if (numMoved > 0)
        // 从 index + 1 位置开始被拷贝，拷贝的起始位置是 index，长度是 numMoved
        System.arraycopy(elementData, index+1, elementData, index, numMoved);
    //数组最后一个位置赋值 null，帮助 GC
    elementData[--size] = null;
}
```

从源码中，我们可以看出，某一个元素被删除后，为了维护数组结构，我们都会把数组后面的元素往前移动，下面动图也演示了其过程：



目录

第1章 基础

01 开篇词：为什么学习本专栏

02 String、Long 源码解析和面试题

03 Java 常用关键字理解

04 Arrays、Collections、Objects 常用方法源码解析

第2章 集合

05 ArrayList 源码解析和设计思路 [最近阅读](#)

06 LinkedList 源码解析

07 List 源码会问哪些面试题

08 HashMap 源码解析

09 TreeMap 和 LinkedHashMap 核心源码解析

10 Map源码会问哪些面试题

11 HashSet、TreeSet 源码解析

12 彰显细节：看集合源码对我们实际工作的帮助和应用

13 差异对比：集合在 Java 7 和 8 有何不同和改进

14 简化工作：Guava Lists Maps 实际工作运用和源码

第3章 并发集合类

15 CopyOnWriteArrayList 源码解析和设计思路

16 ConcurrentHashMap 源码解析和设计思路

17 并发 List、Map源码面试题

18 场景集合：并发 List、Map的应用

如果要自己实现迭代器，实现 java.util.Iterator 类就好了，ArrayList 也是这样做的，我们来看下迭代器的几个总要的参数：

```
int cursor;// 迭代过程中，下一个元素的位置，默认从 0 开始。
int lastRet = -1; // 新增场景：表示上一次迭代过程中，索引的位置；删除场景：为 -1。
int expectedModCount = modCount;// expectedModCount 表示迭代过程中，期望的版本号；m
```

迭代器一般来说有三个方法：

- hasNext 还有没有值可以迭代
- next 如果有值可以迭代，迭代的值是多少
- remove 删除当前迭代的值

我们来分别看下三个方法的源码：

hasNext

```
public boolean hasNext() {
    return cursor != size;//cursor 表示下一个元素的位置，size 表示实际大小，如果两者相等，说明已
}
```

next

```
public E next() {
    //迭代过程中，判断版本号有无被修改，有被修改，抛 ConcurrentModificationException 异常
    checkForComodification();
    //本次迭代过程中，元素的索引位置
    int i = cursor;
    if (i >= size)
        throw new NoSuchElementException();
    Object[] elementData = ArrayList.this.elementData;
    if (i >= elementData.length)
        throw new ConcurrentModificationException();
    // 下一次迭代时，元素的位置，为下一次迭代做准备
    cursor = i + 1;
    // 返回元素值
    return (E) elementData[lastRet = i];
}
// 版本号比较
final void checkForComodification() {
    if (modCount != expectedModCount)
        throw new ConcurrentModificationException();
}
```

从源码中可以看到，next 方法就干了两件事情，第一是检验能不能继续迭代，第二是找到迭代的值，并为下一次迭代做准备（cursor+1）。

remove

```
public void remove() {
    // 如果上一次操作时，数组的位置已经小于 0 了，说明数组已经被删除完了
```



<div>← 慕课专栏</div>	<div>面试官系统精讲Java源码及大厂真题 / 05 ArrayList 源码解析和设计思路</div>
<div>目录</div>	<div><div>checkForComodification();</div><div>try { ArrayList.this.remove(lastRet); cursor = lastRet; // -1 表示元素已经被删除，这里也防止重复删除 lastRet = -1; // 删除元素时 modCount 的值已经发生变化，在此赋值给 expectedModCount // 这样下次迭代时，两者的值是一致的了 expectedModCount = modCount; } catch (IndexOutOfBoundsException ex) { throw new ConcurrentModificationException(); } }</div></div>
<div>第1章 基础</div>	
<div>01 开篇词：为什么学习本专栏</div>	
<div>02 String、Long 源码解析和面试题</div>	
<div>03 Java 常用关键字理解</div>	
<div>04 Arrays、Collections、Objects 常用方法源码解析</div>	
<div>第2章 集合</div>	
<div>05 ArrayList 源码解析和设计思路 <a href="#">最近阅读</a></div>	
<div>06 LinkedList 源码解析</div>	
<div>07 List 源码会问哪些面试题</div>	
<div>08 HashMap 源码解析</div>	
<div>09 TreeMap 和 LinkedHashMap 核心源码解析</div>	
<div>10 Map源码会问哪些面试题</div>	
<div>11 HashSet、TreeSet 源码解析</div>	
<div>12 彰显细节：看集合源码对我们实际工作的帮助和应用</div>	
<div>13 差异对比：集合在 Java 7 和 8 有何不同和改进</div>	
<div>14 简化工作：Guava Lists Maps 实际工作运用和源码</div>	
<div>第3章 并发集合类</div>	
<div>15 CopyOnWriteArrayList 源码解析和设计思路</div>	
<div>16 ConcurrentHashMap 源码解析和设计思路</div>	
<div>17 并发 List、Map源码面试题</div>	
<div>18 场景集合：并发 List、Map的应用</div>	

这里我们需要注意的两点是：

- lastRet = -1 的操作目的，是防止重复删除操作
- 删除元素成功，数组当前 modCount 就会发生变化，这里会把 expectedModCount 重新赋值，下次迭代时两者的值就会一致了

### 2.6 时间复杂度

从我们上面新增或删除方法的源码解析，对数组元素的操作，只需要根据数组索引，直接新增和删除，所以时间复杂度是 O (1)。

### 2.7 线程安全

我们需要强调的是，只有当 ArrayList 作为共享变量时，才会有线程安全问题，当 ArrayList 是方法内的局部变量时，是没有线程安全的问题的。

ArrayList 有线程安全问题的本质，是因为 ArrayList 自身的 elementData、size、modConut 在进行各种操作时，都没有加锁，而且这些变量的类型并非是可见（volatile）的，所以如果多个线程对这些变量进行操作时，可能会有值被覆盖的情况。

类注释中推荐我们使用 Collections#synchronizedList 来保证线程安全，SynchronizedList 是通过在每个方法上面加上锁来实现，虽然实现了线程安全，但是性能大大降低，具体实现源码：

```
public boolean add(E e) {  
    synchronized (mutex) { // synchronized 是一种轻量锁，mutex 表示一个当前 SynchronizedList  
        return c.add(e);  
    }  
}
```

### 总结

本文从 ArrayList 整体架构出发，落地到初始化、新增、扩容、删除、迭代等核心源码实现，我们发现 ArrayList 其实就是围绕底层数组结构，各个 API 都是对数组的操作进行封装，让使用者无需感知底层实现，只需关注如何使用即可。

← 慕课专栏	面试官系统精讲Java源码及大厂真题 / 05 ArrayList 源码解析和设计思路
目录	精选留言 33
第1章 基础	欢迎在这里发表留言，作者筛选后可公开显示
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路 <a href="#">最近阅读</a>	
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	



← 慕课专栏			:三 面试官系统精讲Java源码及大厂真题 / 05 ArrayList 源码解析和设计思路		
目录			回复 2019-12-07 15:58:32		
第1章 基础			文贺 回复 凉凉那个凉凉		
01 开篇词：为什么学习本专栏			下面几个同学回答的都非常好，我理解这可能就是一种抽象的思想，比如我们想要 List 循环时，我们给 List 写 for 循环，想要 Set 循环时，我们给 Set 写 for 循环，如果再来 Map 呢？这时候我们会发现这些循环基本都是重复代码，而且还有一些坑在里面，这时候我们就可以把循环这种行为抽象成迭代器，这样也方便容器的扩展，比如以后新建一个容器后，只要容器实现了迭代器的规范，自然就拥有了了迭代的功能，这就是我们平时工作中经常要到的方法论：先复制粘贴，然后再把公用的抽象，最后经过验证后，把抽象的接口提供给其他程序使用。		
02 String、Long 源码解析和面试题			回复 2019-12-08 13:41:07		
03 Java 常用关键字理解					
04 Arrays、Collections、Objects 常用方法源码解析					
第2章 集合			洗衣粉1		
05 ArrayList 源码解析和设计思路 最近阅读			老师，elementData设置成了transient，那ArrayList是怎么把元素序列化的呢？		
06 LinkedList 源码解析			👍 1 回复 2019-11-02		
07 List 源码会问哪些面试题			文贺 回复 洗衣粉1		
08 HashMap 源码解析			同学你好，这个问题其实自己 debug 一下基本就能发现问题，多多 debug。和序列化工具关系很大，我用的是阿里的 fastjson 进行序列化，debug 在 com.alibaba.fastjson.serializer.ListSerializer 68行左右，会把 List 里面每个值拿出来 append 到 SerializeWriter(可以理解成 StringBuffer的缓冲区) 上，最后输出一定格式的字符串，如果是其他序列化工具的话，可能又是其他的方式了，transient 关键字用途是好的，但序列化方式很多，两者没有太大的关系。		
09 TreeMap 和 LinkedHashMap 核心源码解析			回复 2019-11-04 10:44:10		
10 Map源码会问哪些面试题			所相虚妄 回复 洗衣粉1		
11 HashSet、TreeSet 源码解析			他自己实现了那个接口啊		
12 彰显细节：看集合源码对我们实际工作的帮助和应用			回复 2019-11-05 12:39:26		
13 差异对比：集合在 Java 7 和 8 有何不同和改进			wwwwwwwwei 回复 洗衣粉1		
14 简化工作：Guava Lists Maps 实际工作运用和源码			ArrayList里面的readObject和writeObject方法了解一下？		
第3章 并发集合类			回复 2019-11-16 11:30:53		
15 CopyOnWriteArrayList 源码解析和设计思路					
16 ConcurrentHashMap 源码解析和设计思路			Sicimike		
17 并发 List、Map源码面试题			大佬，ArrayList中存储数据的数组为啥是Object[]，而不是E[]。是因为泛型在1.5才引入，而ArrayList在1.2就有了吗？		
18 场景集合：并发 List、Map的应用			👍 0 回复 2019-10-28		
			文贺 回复 Sicimike		
			你分析的很有道理的。		
			回复 2019-10-31 12:55:27		
			少为 回复 Sicimike		
			也许还有另外一个原因，Java是不支持创建泛型数组的。当然可以创建Object数组然后强转为E[]。		
			回复 2019-11-20 19:31:58		
			树上有_云		
			size 不是当前数组的大小吧，而是元素的个数		
			👍 0 回复 2019-10-24		

<div>← 慕课专栏</div> <div>面试官系统精讲Java源码及大厂真题 / 05 ArrayList 源码解析和设计思路</div>		
目录	回复	2019-12-05 21:48:36
第1章 基础		
01 开篇词：为什么学习本专栏		
02 String、Long 源码解析和面试题		
03 Java 常用关键字理解		
04 Arrays、Collections、Objects 常用方法源码解析		
第2章 集合		
05 ArrayList 源码解析和设计思路 <a href="#">最近阅读</a>		
06 LinkedList 源码解析		
07 List 源码会问哪些面试题		
08 HashMap 源码解析		
09 TreeMap 和 LinkedHashMap 核心源码解析		
10 Map源码会问哪些面试题		
11 HashSet、TreeSet 源码解析		
12 彰显细节：看集合源码对我们实际工作的帮助和应用		
13 差异对比：集合在 Java 7 和 8 有何不同和改进		
14 简化工作：Guava Lists Maps 实际工作运用和源码		
第3章 并发集合类		
15 CopyOnWriteArrayList 源码解析和设计思路		
16 ConcurrentHashMap 源码解析和设计思路		
17 并发 List、Map源码面试题		
18 场景集合：并发 List、Map的应用		

南船座

int oldCapacity = elementData.length; // oldCapacity >> 1 是把 oldCapacity 除以 2 的意思 int newCapacity = oldCapacity + (oldCapacity >> 1); // 如果扩容后的值 < 我们的期望值，扩容后的值就等于我们的期望值 if (newCapacity - minCapacity < 0) newCapacity = minCapacity; newCapacity已经是oldCapacity除以2再加上oldCapacity了 那无论如何都是不存在的newCapacity小于oldCapacity if (newCapacity - minCapacity < 0) newCapacity = minCapacity; 为什么还要这么写？

1

回复

2019-10-21

文贺 回复 南船座

同学你好，你可以看下 allAll 方法，这个方法引起的扩容，minCapacity 有可能非常大，甚至大于 minCapacity，就会出现 newCapacity - minCapacity 的情况。

回复

2019-10-21 22:03:54

魅影劲 回复 南船座

参考07节List源码会问那些面试题 1.2.3 就有这种CASE

回复

2019-11-18 16:12:47

喵喵喵111

向末尾插入数据的时候，确实是o（1），如果刚好在需要扩容的临界点进行添加和删除，那么不就是o(n)了么

2

回复

2019-10-19

文贺 回复 喵喵喵111

是的，正好碰到扩容时候是的。

回复

2019-10-21 22:00:13

魅影劲 回复 喵喵喵111

根据均摊时间复杂度分析,当我们添加到最后一个元素时，如果此时再往里添加一个元素，那么就将进行数组的扩容操作。这次扩容的算法复杂度为O(n),此时我们就要用到均摊复杂度分析法，前面n次操作耗时间总共为n,第n+1次操作耗时间为n,相当于执行n+1次操作耗费的时间为2 n,那么平均来看，我们每次操作其实耗费的时间为2，他仍然是一个O(1)级别的算法。在这里我把一次线性操作(第n+1次)的复杂度均摊到前面n次操作中。

回复

2019-11-18 15:23:12

喵喵喵111 回复 魅影劲

那如果我在扩容以后立马缩容。就在这个临界点来回添加删除，那时间复杂度不就变成了on了么

回复

2019-11-18 16:36:02

点击展开后面 2 条

喵喵喵111

老师，如果数组刚好处于扩容的临界点，这时候进行数组的增加，再减少，每次的复杂度都会为o(n)，并不是所说的o(1)呀

2

回复

2019-10-19

← 慕课专栏	面试官系统精讲Java源码及大厂真题 / 05 ArrayList 源码解析和设计思路
目录	
第1章 基础	<a href="#">点击展开剩余评论</a>
01 开篇词：为什么学习本专栏	
02 String、Long 源码解析和面试题	千学不如一看，千看不如一练
03 Java 常用关键字理解	
04 Arrays、Collections、Objects 常用方法源码解析	
第2章 集合	
05 ArrayList 源码解析和设计思路	<a href="#">最近阅读</a>
06 LinkedList 源码解析	
07 List 源码会问哪些面试题	
08 HashMap 源码解析	
09 TreeMap 和 LinkedHashMap 核心源码解析	
10 Map源码会问哪些面试题	
11 HashSet、TreeSet 源码解析	
12 彰显细节：看集合源码对我们实际工作的帮助和应用	
13 差异对比：集合在 Java 7 和 8 有何不同和改进	
14 简化工作：Guava Lists Maps 实际工作运用和源码	
第3章 并发集合类	
15 CopyOnWriteArrayList 源码解析和设计思路	
16 ConcurrentHashMap 源码解析和设计思路	
17 并发 List、Map源码面试题	
18 场景集合：并发 List、Map的应用	