#### [java集合框架之ArrayList深度解析（二）](http://www.spring4all.com/article/1050)

#### ****摘要：****

前面部分讲解了ArrayList的无参构造器和add(E e)方法，本章接着讲解指定容量大小的构造器和其它方法的详细使用，由于这个集合框架的使用比较广泛，相信大家都有很多的理解，如有讲解错误还希望各位评论中指点出来。

##### ****ArrayList指定容量大小构造器****

*/\*\**

*\* 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) {

*//默认无参构造器使用的DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA，它们都是Object[]*

this.elementData = EMPTY\_ELEMENTDATA;

} else {

throw new IllegalArgumentException("Illegal Capacity: "+

initialCapacity);

}

}

着重分析一下ArrayList中核心对象数组Object[] 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*

#### ****transient与elementData****

首先transient是什么？，为什么用transient修饰符？带着这2个问题继续下面的探究吧。。。。。。。  
transient用来表示一个域不是该对象序行化的一部分，当一个对象被序行化的时候，transient修饰的变量的值是不包括在序行化的表示中的。但是ArrayList又是可序行化的类，elementData是ArrayList具体存放元素的成员，用transient来修饰elementData，岂不是反序列化后的ArrayList丢失了原先的元素？  
仔细研读源码会发现以下2个方法：

*/\*\**

*\* 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 size as capacity for behavioural compatibility with clone()*

s.writeInt(size);

*// Write out all elements in the proper order.*

*//注意这个地方size的大小和elementData的长度关系，它们之间是size&lt;=elementData.length，什么时候相等呢？*

*//那就是刚好size+1=容量大小从而不需要扩容，然后执行elementData[size++] = e;*

for (int i=0; i&lt;size; i++) {<br>

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 {

elementData = EMPTY\_ELEMENTDATA;

*// Read in size, and any hidden stuff*

s.defaultReadObject();

*// Read in capacity*

s.readInt(); *// ignored*

if (size > 0) {

*// be like clone(), allocate array based upon size not capacity*

*//刚好分配序列化之前容量中变化值长度大小，通俗讲就是实际容量中数据的个数*

int capacity = calculateCapacity(elementData, size);

SharedSecrets.getJavaOISAccess().checkArray(s, Object[].class, capacity);

ensureCapacityInternal(size);

Object[] a = elementData;

*// Read in all elements in the proper order.*

for (int i=0; i&lt;size; i++) {

a[i] = s.readObject();

}

}

}

**综上所述：**

ArrayList在序列化的时候会调用writeObject，直接将size和element写入ObjectOutputStream；反序列化时调用readObject，从ObjectInputStream获取size和element，再恢复到elementData。  
为什么不直接用elementData来序列化，而采用上诉的方式来实现序列化呢？原因在于elementData是一个缓存数组，它通常会预留一些容量，等容量不足时再扩充容量，那么有些空间可能就没有实际存储元素，采用上诉的方式来实现序列化时，就可以保证只序列化实际存储的那些元素，而不是整个数组，从而节省空间和时间。

#### ****modCount分析****

*/\*\**

*\* The number of times this list has been <i>structurally modified</i>.*

*\* Structural modifications are those that change the size of the*

*\* list, or otherwise perturb it in such a fashion that iterations in*

*\* progress may yield incorrect results.*

*\**

*\* <p>This field is used by the iterator and list iterator implementation*

*\* returned by the {@code iterator} and {@code listIterator} methods.*

*\* If the value of this field changes unexpectedly, the iterator (or list*

*\* iterator) will throw a {@code ConcurrentModificationException} in*

*\* response to the {@code next}, {@code remove}, {@code previous},*

*\* {@code set} or {@code add} operations. This provides*

*\* <i>fail-fast</i> behavior, rather than non-deterministic behavior in*

*\* the face of concurrent modification during iteration.*

*\**

*\* <p><b>Use of this field by subclasses is optional.</b> If a subclass*

*\* wishes to provide fail-fast iterators (and list iterators), then it*

*\* merely has to increment this field in its {@code add(int, E)} and*

*\* {@code remove(int)} methods (and any other methods that it overrides*

*\* that result in structural modifications to the list). A single call to*

*\* {@code add(int, E)} or {@code remove(int)} must add no more than*

*\* one to this field, or the iterators (and list iterators) will throw*

*\* bogus {@code ConcurrentModificationExceptions}. If an implementation*

*\* does not wish to provide fail-fast iterators, this field may be*

*\* ignored.*

*\*/*

protected transient int modCount = 0;

**译文：**  
该字段表示list结构上被修改的次数。结构上的修改指的是那些改变了list的长度大小或者使得遍历过程中产生不正确的结果的其它方式。  
该字段被Iterator以及list Iterator的实现类所使用，如果该值被意外更改，Iterator或者list Iterator 将抛出ConcurrentModificationException异常，  
这是jdk在面对迭代遍历的时候为了避免不确定性而采取的快速失败原则。  
子类对此字段的使用是可选的，如果子类希望支持快速失败，只需要覆盖该字段相关的所有方法即可。单线程调用不能添加删除Iterator正在遍历的对象，  
否则将可能抛出ConcurrentModificationException异常，如果子类不希望支持快速失败，该字段可以直接忽略。

##### ****例子如下：****

public static void main(String[] args) {

List&lt;String> list = new ArrayList();

list.add("d");

list.add("q");

list.add("q");

list.add("z");

list.add("j");

Iterator it = list.iterator();

int i = 0;

while(it.hasNext()){

if(i==4){

*//list.remove(it.next());*

it.remove();*// 如果用list.remove(it.next());会报异常*

}

System.out.println("第"+i+"个元素"+it.next());

i++ ;

}

System.out.println("----------------");

Iterator it2 = list.iterator();

while(it2.hasNext()){

System.out.println(it2.next());

}

}

**注意：**  
如果用list.remove(it.next());会报ConcurrentModificationException异常，原因参上。

另：注意it.remove()删除的是最近的一次it.next()获取的元素，而不是当前iterator中游标指向的元素！！  
因此，本例中i==4时，删除的其实是z，而不是j，这很容易被忽视或者误解。如果想删掉j正确操作是先调用it.next()获取到具体元素，再判断；而且由于调用了it.next(),此时游标已经指向我们期望删除的值了。想直接数数字进行删除，在这里会容易出错误。后续章节会深入讲解iterator迭代器源码。

### ****总结：****

本篇文章主要分析了ArrayListzh中的3个重要的成员变量，elementdata和modCount和随便提及的size,总的来说可以归结为下面3个部分成员变量  
elementData：底层Object[]  
size：ArrayList中元素的实际个数  
modCount：记录ArrayList的修改次数

##### ****参考文献****

[Collections Framework Overview](https://docs.oracle.com/javase/6/docs/technotes/guides/collections/overview.html)  
[The For-Each Loop](https://docs.oracle.com/javase/1.5.0/docs/guide/language/foreach.html)

##### ****参考本系列文章****

[Java 集合框架之概览](http://www.spring4all.com/article/1042)  
[java集合框架之ArrayList深度解析（一）](http://www.spring4all.com/article/1045)  
[java集合框架之ArrayList深度解析（二）](http://www.spring4all.com/article/1050)  
[java集合框架之ArrayList深度解析（三）](http://www.spring4all.com/article/1055)  
[java集合框架之ArrayList深度解析（四）](http://www.spring4all.com/article/1056)