# ArrayLIst源码剖析

Jdk1.8中的ArrayList为例

|  |
| --- |
| **默认容量**  private static final int DEFAULT\_CAPACITY = 10;  **共享空常量数组**  private static final Object[] EMPTY\_ELEMENTDATA = {};  **共享空常量数组（与**EMPTY\_ELEMENTDATA的区别在于第一个元素被加进来的时候，知道如何扩展**）**  private static final Object[] DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA = {};      **存放数组的缓存容量，不可序列化，ArrayList底层是使用数组来实现的**  transient Object[] elementData;  **记录list中存放的元素个数**  private int size;  **指定扩展容量的构造器，参数>0按照指定容量扩展数组，参数=0用空的常量数组对象EMPTY\_ELEMENTDATA，参数<0时，抛异常不合法的容量**  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);  }  }  **无参构造器**  public ArrayList() {  this.elementData = DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA;  }  构造一个初始容量为 10 的空列表，当我们使用无参构造器new一个对象时，底层数组就是直接用长度为10的空常量数组DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA进行实例化；这个在添加第一个元素时调用add(E)或者 add(int, E)时候就会调用下面方法    Grow函数使用minCapacity进行扩容，此时的minCapacity就是初始值10；，grow函数后面详讲。  **使用集合toArray返回object[]，并赋值给elementData，如果collection长度不为零，判断toArray方法是否真的返回了**Object[].class，如果没有就，使用Arrays.copyOf将elementData的副本拷贝为Object[].class类型，复制给**elementData，如果为空，空的常量数组对象EMPTY\_ELEMENTDATA赋给了elementData；**  public ArrayList(Collection<? extends E> c) {  elementData = c.toArray();  if ((size = elementData.length) != 0) {  if (elementData.getClass() != Object[].class)  elementData = Arrays.copyOf(elementData, size, Object[].class);  } else {  // replace with empty array.  this.elementData = EMPTY\_ELEMENTDATA;  }  }  **扩容方法**  private void grow(int minCapacity) {  // overflow-conscious code  int oldCapacity = elementData.length;  int newCapacity = oldCapacity + (oldCapacity >> 1);  **每次新增容量为旧容量的一半**  if (newCapacity - minCapacity < 0)  newCapacity = minCapacity;//**扩容扩的太小，就等于**minCapacity  if (newCapacity - MAX\_ARRAY\_SIZE > 0)  newCapacity = hugeCapacity(minCapacity);// **扩的太大，大于输出最大值就取Integer.MAX\_VALUE，否则等于最大值**  //将原来的数组中的元素复制扩展到大小为newCapacity的新数组中，并返回这个新数组  elementData = Arrays.copyOf(elementData, newCapacity);  /\*  public static <T> T[] copyOf(T[] original, int newLength) {  return (T[]) copyOf(original, newLength, original.getClass());  }  public static <T,U> T[] copyOf(U[] original, int newLength, Class<? extends T[]> newType) {  @SuppressWarnings("unchecked")  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;  }\*/  }  **//大于数组最大值就取Integer.MAX\_VALUE，否则等于最大值**  private static int hugeCapacity(int minCapacity) {  if (minCapacity < 0) // overflow  throw new OutOfMemoryError();  return (minCapacity > MAX\_ARRAY\_SIZE) ?  Integer.MAX\_VALUE :  MAX\_ARRAY\_SIZE;  }  **返回此ArrayList实例的浅拷贝。 （元素本身不被复制。）**  public Object clone() {  try {  ArrayList<?> v = (ArrayList<?>) super.clone();  v.elementData = Arrays.copyOf(elementData, size);  v.modCount = 0;  return v;  } catch (CloneNotSupportedException e) {  // this shouldn't happen, since we are Cloneable  throw new InternalError(e);  }  }  **直接使用数组随机访问的特点**  public E get(int index) {  rangeCheck(index);//**有效性检查**  return elementData(index);  }  //**返回旧值，设置新值**  public E set(int index, E element) {  rangeCheck(index);//有效性检查  E oldValue = elementData(index);  elementData[index] = element;  return oldValue;  }  **前面已经介绍过了**  public boolean add(E e) {  ensureCapacityInternal(size + 1); // Increments modCount!!  elementData[size++] = e;  return true;  }    public void add(int index, E element) {  rangeCheckForAdd(index);  //添加修改次数，**判断是否需要扩张**  ensureCapacityInternal(size + 1); // Increments modCount!!  //移动元素以备插入数据  System.arraycopy(elementData, index, elementData, index + 1,  size - index);  elementData[index] = element;  size++;  }  //移除**指定位置的元素，**  public E remove(int index) {  rangeCheck(index);  modCount++;  E oldValue = elementData(index);//保存要删除的元素  int numMoved = size - index - 1;  if (numMoved > 0)//数组移位  System.arraycopy(elementData, index+1, elementData, index,  numMoved);  elementData[--size] = null; // clear to let GC do its work  return oldValue;  }  **无论是指定对象o是否为null，都是在ArrayList中找到与此第一个相等的元素的位置，然后调用fastRemove(index)来进行移除；如果没有找打指定对象o的位置，则返回false，表示没有移除成功。**  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;  }  **与remove基本没差，少了有效性检查以及没有返回删除的值。**  private void fastRemove(int index) {  modCount++;  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  } |