　http://www.cnblogs.com/whgk/p/6079212.html

　序言

　　　　　　　　　　第一次看源码，借鉴的是这位博主的文章：[http://blog.csdn.net/csh624366188/article/details/6896656](http://blog.csdn.net/csh624366188/article/details/6896656" \o "http://blog.csdn.net/csh624366188/article/details/6896656" \t "http://www.cnblogs.com/whgk/p/_blank)  个人觉得他写的一系列文章很好，稍微有点深度，他是一个农村的，我也是，佩服他的坚持，希望自己也能坚持下去，他写的这篇文章还是在2011年，那时候他的jdk版本估计也才4到5左右，现在jdk已经到8了，我查看的源码是jdk7，有那么一点点不同，不过大体上是没什么改变的。在阅读的过程中，自己也慢慢的适应去看api文档，全英文版的，第一次看完了一个方法，花了我大概30分钟的时间，但是感觉收获满满，下面我会一一介绍的，用的是有道词典，直接官网可以下载，个人感觉还是很不错的，有道确实越来越好了，给出网址([http://cidian.youdao.com/?keyfrom=dict2.index](http://cidian.youdao.com/?keyfrom=dict2.index" \o "http://cidian.youdao.com/?keyfrom=dict2.index" \t "http://www.cnblogs.com/whgk/p/_blank))，自己又需要的可以去下载。

　　　　　　　　　　这篇文章个人感觉帮助我很多。感兴趣的同学也可以去看看。

　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　----WH

一、查看源码的方法

　　　　　　　自己也是第一次看源码，从上面介绍的那位博主哪里学到的，以后再看源码多的话，应该还会滋生出更多的经验来，设计模式要懂，要知道大概的层次结构。

　　　　　　　　　　1、看继承结构

　　　　　　　　　　　　　　　看这个类的层次结构，处于一个什么位置，可以在自己心里有个大概的了解。

　　　　　　　　　　2、看构造方法

　　　　　　　　　　　　　　在构造方法中，看做了哪些事情，跟踪方法中里面的方法

　　　　　　　　　   3、看常用的方法

　　　　　　　　　　　　　　跟构造方法一样，这个方法实现功能是如何实现的

二、ArrayList源代码

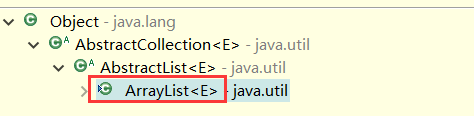
　　　　　　1、继承结构和层次关系

 　　　　　　　　　　　首先我们来看一下ArrayList的继承结构，

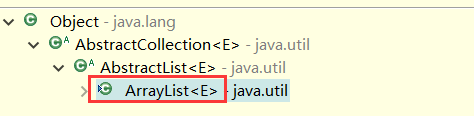
　　　　　　　　　　　　　　　ArrayList extends AbstractList

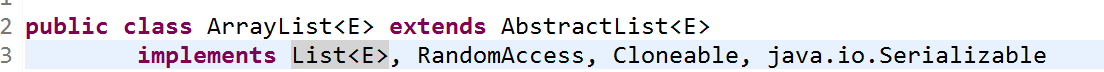
　　　　　　　　　　　　　　　AbstractList extends AbstractCollection

　　　　　　　　　　　　　　　所有类都继承Object  所以ArrayList的继承结构就是下图这样

　　　　　　　　　1、为什么要先继承AbstractList，而让AbstractList先实现List<E>？而不是让ArrayList直接实现List<E>？

　　　　　　　　　　　　　　　　　　这里是有一个思想，接口中全都是抽象的方法，而抽象类中可以有抽象方法，还可以有具体的实现方法，正是利用了这一点，让AbstractList是实现接口中一些通用的方法，而具体的类，如ArrayList就继承这个AbstractList类，拿到一些通用的方法，然后自己在实现一些自己特有的方法，这样一来，让代码更简洁，就继承结构最底层的类中通用的方法都抽取出来，先一起实现了，减少重复代码。所以一般看到一个类上面还有一个抽象类，应该就是这个作用





 　　　　　　　　　　　在来看看ArrayList实现了哪些接口？

　　　　　　　　　　　　　　　　1、List<E>接口：我们会出现这样一个疑问，在查看了ArrayList的父类AbstractList也实现了List<E>接口，那为什么子类ArrayList还是去实现一遍呢？这是想不通的地方，所以我就去查资料，有的人说是为了查看代码方便，使观看者一目了然，说法不一，但每一个让我感觉合理的，但是在stackOverFlow中找到了答案，这里其实很有趣，网址贴出来 [http://stackoverflow.com/questions/2165204/why-does-linkedhashsete-extend-hashsete-and-implement-sete](http://stackoverflow.com/questions/2165204/why-does-linkedhashsete-extend-hashsete-and-implement-sete" \o "http://stackoverflow.com/questions/2165204/why-does-linkedhashsete-extend-hashsete-and-implement-sete" \t "http://www.cnblogs.com/whgk/p/_blank)开发这个collection 的作者Josh说。这其实是一个mistake，因为他写这代码的时候觉得这个会有用处，但是其实并没什么用，但因为没什么影响，就一直留到了现在。

　　　　　　　　　　　　　　　　2、RandomAccess接口：这个是一个标记性接口，通过查看api文档，它的作用就是用来快速随机存取，有关效率的问题，在实现了该接口的话，那么使用普通的for循环来遍历，性能更高，例如arrayList。而没有实现该接口的话，使用Iterator来迭代，这样性能更高，例如linkedList。所以这个标记性只是为了让我们知道我们用什么样的方式去获取数据性能更好，可以参考这篇博文，[http://blog.csdn.net/keda8997110/article/details/8635005](http://blog.csdn.net/keda8997110/article/details/8635005" \o "http://blog.csdn.net/keda8997110/article/details/8635005" \t "http://www.cnblogs.com/whgk/p/_blank)

强调:

JDK中推荐的是对List集合尽量要实现RandomAccess接口

如果集合类是RandomAccess的实现，则尽量用for(int i = 0; i < size; i++) 来遍历而不要用Iterator迭代器来遍历，在效率上要差一些。反过来，如果List是Sequence List，则最好用迭代器来进行迭代。

JDK中说的很清楚，在对List特别是Huge size的List的遍历算法中，要尽量来判断是属于RandomAccess(如ArrayList)还是Sequence List (如LinkedList），因为适合RandomAccess List的遍历算法，用在Sequence List上就差别很大，常用的作法就是：  
    要作一个判断：  
    if (list instance of RandomAccess) {  
        for(int m = 0; m < list.size(); m++){}  
    }else{  
        Iterator iter = list.iterator();  
        while(iter.hasNext()){}  
    }

　　　　　　　　　　　　　　　　3、Cloneable接口：实现了该接口，就可以使用Object.Clone()方法了。

　　　　　　　　　　　　　　　　4、Serializable接口：实现该序列化接口，表明该类可以被序列化，什么是序列化？简单的说，就是能够从类变成字节流传输，然后还能从字节流变成原来的类。

private static final int DEFAULT\_CAPACITY = 10;

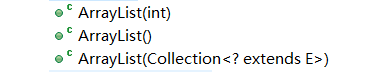
private static final Object[] EMPTY\_ELEMENTDATA = {};

private static final Object[] DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA = {}

transient Object[] elementData;

　　　　　　2、构造方法

 　　　　　　　　　　三个构造方法



　　　　　　　　　　1、无参构造方法：(原谅我一直不知道能这样贴代码，搞得之前一直都是发截图。)

[IMG_259](http://www.cnblogs.com/whgk/p/javascript:void(0);)

1 /\*\*  
2     \* Constructs an empty list with an initial capacity of ten.　　这里就说明了默认会给10的大小，所以说一开始arrayList的容量是10.  
3     \*/  
　　　　//ArrayList中储存数据的其实就是一个数据，这个数组就是elementData，在123行定义的 private transient Object[] elementData;  
4　　　 public ArrayList() {

5 super(); //调用父类中的无参构造方法，父类中的是个空的构造方法

6 this.elementData = EMPTY\_ELEMENTDATA;//EMPTY\_ELEMENTDATA：是个空的Object[]， 将elementData初始化，elementData也是个Object[]类型。空的Object[]会给默认大小10，等会会解释什么时候赋值的 并没有在这里初始化

7 }

[IMG_260](http://www.cnblogs.com/whgk/p/javascript:void(0);)

 this.elementData = DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA; 不一样的是默认的空的数组也是没有初始化的

2、有参构造方法1

　　　　　　　　　/\*\*

\* 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) {

super(); //父类中空的构造方法

if (initialCapacity < 0) //判断如果自定义大小的容量小于0，则报下面这个非法数据异常

throw new IllegalArgumentException("Illegal Capacity: "+

initialCapacity);

this.elementData = new Object[initialCapacity]; //将自定义的容量大小当成初始化elementData的大小 初始化了

}

　　　　　　　　　3、有参构造方法2(不常用)

[IMG_263](http://www.cnblogs.com/whgk/p/javascript:void(0);)

//这个构造方法不常用，举个例子就能明白什么意思

/\*

Strudent entends Person

ArrayList<Person>、 Person这里就是泛型

我还有一个Collection<Student>、由于这个Student继承了Person，那么根据这个构造方法，我就可以把这个Collection<Student>转换为ArrayList<Sudent>这就是这个构造方法的作用

\*/

public ArrayList(Collection<? extends E> c) {

elementData = c.toArray(); //转换为数组

size = elementData.length; //数组中的数据个数

// c.toArray might (incorrectly) not return Object[] (see 6260652)

if (elementData.getClass() != Object[].class) //每个集合的toarray()的实现方法不一样，所以需要判断一下，如果不是Object[].class类型，那么久需要使用ArrayList中的方法去改造一下。

elementData = Arrays.copyOf(elementData, size, Object[].class);

}

[IMG_264](http://www.cnblogs.com/whgk/p/javascript:void(0);)

 　　总结一下：arrayList的构造方法就做一件事情，就是初始化一下储存数据的容器，其实本质上就是一个数组，在其中就叫elementData。其他什么也不做了。

　　　　　　3、常用方法

 　　　　　　　　 　　add()方法有两个

IMG_265

　　　　　　　　　　　　　　boolean add(E)；//默认直接在末尾添加元素。

[IMG_266](http://www.cnblogs.com/whgk/p/javascript:void(0);)

/\*\*

\* Appends the specified element to the end of this list.添加一个特定的元素到list的末尾。

\*

\* @param e element to be appended to this list

\* @return <tt>true</tt> (as specified by {@link Collection#add})

\*/

public boolean add(E e) {

//确定内部容量是否够了，size是数组中数据的个数，因为要添加一个元素，所以size+1，先判断size+1的这个个数数组能否放得下，就在这个方法中去判断是否数组.length是否够用了。

ensureCapacityInternal(size + 1); // Increments modCount!!

//在数据中正确的位置上放上元素e，并且size++

elementData[size++] = e;

return true;

}

[IMG_267](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　　　　　　　　ensureCapacityInternal(xxx);

[IMG_268](http://www.cnblogs.com/whgk/p/javascript:void(0);)

private void ensureCapacityInternal(int minCapacity) {

if (elementData == EMPTY\_ELEMENTDATA（DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA）) { //看，判断初始化的elementData是不是空的数组，也就是没有长度

//因为如果是空的话，minCapacity=size+1；其实就是等于1，空的数组没有长度就存放不了，所以就将minCapacity变成10，也就是默认大小，但是带这里，还没有真正的初始化这个elementData的大小。 默认的数组是没有初始化的是空的数组

minCapacity = Math.max(DEFAULT\_CAPACITY, minCapacity);

}

//确认实际的容量，上面只是将minCapacity=10，这个方法就是真正的判断elementData是否够用 ensureExplicitCapacity(minCapacity);

}

　　ensureExplicitCapacity(xxx)；

private void ensureExplicitCapacity(int minCapacity) {

modCount++;

// overflow-conscious code

//minCapacity如果大于了实际elementData的长度，那么就说明elementData数组的长度不够用，不够用那么就要增加elementData的length。这里有的同学就会模糊minCapacity到底是什么呢，这里给你们分析一下

/\*第一种情况：由于elementData初始化时是空的数组，那么第一次add的时候，minCapacity=size+1；也就minCapacity=1，在上一个方法(确定内部容量ensureCapacityInternal)就会判断出是空的数组，就会给

　　将minCapacity=10，到这一步为止，还没有改变elementData的大小，

　第二种情况：elementData不是空的数组了，那么在add的时候，minCapacity=size+1；也就是minCapacity代表着elementData中增加之后的实际数据个数，拿着它判断elementData的length是否够用，如果length

不够用，那么肯定要扩大容量，不然增加的这个元素就会溢出。

\*/

if (minCapacity - elementData.length > 0)

//arrayList能自动扩展大小的关键方法就在这里了

grow(minCapacity);

}

grow(xxx); arrayList核心的方法，能扩展数组大小的真正秘密。

private void grow(int minCapacity) {

// overflow-conscious code

int oldCapacity = elementData.length; //将扩充前的elementData大小给oldCapacity

int newCapacity = oldCapacity + (oldCapacity >> 1);//newCapacity就是1.5倍的oldCapacity

if (newCapacity - minCapacity < 0)//这句话就是适应于elementData就空数组的时候，length=0，那么oldCapacity=0，newCapacity=0，所以这个判断成立，在这里就是真正的初始化elementData的大小了，就是为10.前面的工作都市准备工作。

newCapacity = minCapacity;

//MAX\_ARRAY\_SIZE =Integer.MAX\_VALUE - 8

if (newCapacity - MAX\_ARRAY\_SIZE > 0)//如果newCapacity超过了最大的容量限制，就调用hugeCapacity，也就是将能给的最大值给newCapacity

newCapacity = hugeCapacity(minCapacity);

// minCapacity is usually close to size, so this is a win:

//新的容量大小已经确定好了，就copy数组，改变容量大小咯。

elementData = Arrays.copyOf(elementData, newCapacity); 初始化

}

hugeCapacity()

//这个就是上面用到的方法，很简单，就是用来赋最大值。

private static int hugeCapacity(int minCapacity) {

if (minCapacity < 0) // overflow

throw new OutOfMemoryError();

//如果minCapacity都大于MAX\_ARRAY\_SIZE，那么就Integer.MAX\_VALUE返回，反之将MAX\_ARRAY\_SIZE返回。因为maxCapacity是三倍的minCapacity，可能扩充的太大了，就用minCapacity来判断了。

//Integer.MAX\_VALUE:2147483647 MAX\_ARRAY\_SIZE：2147483639 也就是说最大也就能给到第一个数值。还是超过了这个限制，就要溢出了。相当于arraylist给了两层防护。

return (minCapacity > MAX\_ARRAY\_SIZE) ?

Integer.MAX\_VALUE :

MAX\_ARRAY\_SIZE;

}

　void add(int，E)；在特定位置添加元素，也就是插入元素

public void add(int index, E element) {

rangeCheckForAdd(index);//检查index也就是插入的位置是否合理。

//跟上面的分析一样，具体看上面

ensureCapacityInternal(size + 1); // Increments modCount!!

//这个方法就是用来在插入元素之后，要将index之后的元素都往后移一位，

System.arraycopy(elementData, index, elementData, index + 1,

size - index);

//在目标位置上存放元素

elementData[index] = element;

size++;//size增加1

}

rangeCheckForAdd(index)//检查index也就是插入的位置是否合理。

private void rangeCheckForAdd(int index) {

if (index > size || index < 0) //插入的位置肯定不能大于size 和小于0//如果是，就报这个越界异常

throw new IndexOutOfBoundsException(outOfBoundsMsg(index));

}

System.arraycopy(...)：就是将elementData在插入位置后的所有元素往后面移一位。查看api文档

public static void arraycopy(Object src,

int srcPos,

Object dest,

int destPos,

int length)

src：源对象

srcPos：源对象对象的起始位置

dest：目标对象

destPost：目标对象的起始位置

length：从起始位置往后复制的长度。

//这段的大概意思就是解释这个方法的用法，复制src到dest，复制的位置是从src的srcPost开始，到srcPost+length-1的位置结束，复制到destPost上，从destPost开始到destPost+length-1的位置上，

Copies an array from the specified source array, beginning at the specified position, to the specified position of the destination array. A subsequence of array components are copied from

the source array referenced by src to the destination array referenced by dest. The number of components copied is equal to the length argument. The components at positions srcPos through srcPos+length-1

in the source array are copied into positions destPos through destPos+length-1, respectively, of the destination array.

//告诉你复制的一种情况，如果A和B是一样的，那么先将A复制到临时数组C，然后通过C复制到B，用了一个第三方参数

If the src and dest arguments refer to the same array object, then the copying is performed as if the components at positions srcPos through srcPos+length-1 were first copied to

a temporary array with length components and then the contents of the temporary array were copied into positions destPos through destPos+length-1 of the destination array.

//这一大段，就是来说明会出现的一些问题，NullPointerException和IndexOutOfBoundsException 还有ArrayStoreException 这三个异常出现的原因。

If dest is null, then a NullPointerException is thrown.

If src is null, then a NullPointerException is thrown and the destination array is not modified.

Otherwise, if any of the following is true, an ArrayStoreException is thrown and the destination is not modified:

The src argument refers to an object that is not an array.

The dest argument refers to an object that is not an array.

The src argument and dest argument refer to arrays whose component types are different primitive types.

The src argument refers to an array with a primitive component type and the dest argument refers to an array with a reference component type.

The src argument refers to an array with a reference component type and the dest argument refers to an array with a primitive component type.

Otherwise, if any of the following is true, an IndexOutOfBoundsException is thrown and the destination is not modified:

The srcPos argument is negative.

The destPos argument is negative.

The length argument is negative.

srcPos+length is greater than src.length, the length of the source array.

destPos+length is greater than dest.length, the length of the destination array.

//这里描述了一种特殊的情况，就是当A的长度大于B的长度的时候，会复制一部分，而不是完全失败。

Otherwise, if any actual component of the source array from position srcPos through srcPos+length-1 cannot be converted to the component type of the destination array by assignment conversion, an ArrayStoreException is thrown.

In this case, let k be the smallest nonnegative integer less than length such that src[srcPos+k] cannot be converted to the component type of the destination array; when the exception is thrown, source array components from positions

srcPos through srcPos+k-1 will already have been copied to destination array positions destPos through destPos+k-1 and no other positions of the destination array will have been modified. (Because of the restrictions already itemized,

this paragraph effectively applies only to the situation where both arrays have component types that are reference types.)

//这个参数列表的解释，一开始就说了，

Parameters:

src - the source array.

srcPos - starting position in the source array.

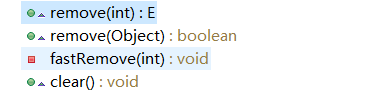
dest - the destination array.

destPos - starting position in the destination data.

length - the number of array elements to be copied.

删除方法有5个

　　　　　　　　　　　　挑几个特点的讲解。其实都类似，其中fastRemove(int)方法是private的，是提供给remove(Object)这个方法用的。

IMG_257IMG_258

　　　　　　　　　　　　remove(int)：通过删除指定位置上的元素

public E remove(int index) {

rangeCheck(index);//检查index的合理性

modCount++;//这个作用很多，比如用来检测快速失败的一种标志。什么是快速失败请看我的每日面试题四中的解答。

E oldValue = elementData(index);//通过索引直接找到该元素

int numMoved = size - index - 1;//计算要移动的位数。

if (numMoved > 0)

//这个方法也已经解释过了，就是用来移动元素的。

System.arraycopy(elementData, index+1, elementData, index,

numMoved);

//将--size上的位置赋值为null，让gc(垃圾回收机制)更快的回收它。

elementData[--size] = null; // clear to let GC do its work

//返回删除的元素。

return oldValue;

}

remove(Object)：这个方法可以看出来，arrayList是可以存放null值得。

//感觉这个不怎么要分析吧，都看得懂，就是通过元素来删除该元素，就依次便利，如果有这个元素，就将该元素的索引传给fastRemobe(index)，使用这个方法来删除该元素，

//fastRemove(index)方法的内部跟remove(index)的实现几乎一样，这里最主要是知道arrayList可以存储null值

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;

}

　clear()：将elementData中每个元素都赋值为null，等待垃圾回收将这个给回收掉，所以叫clear

public void clear() {

modCount++;

// clear to let GC do its work

for (int i = 0; i < size; i++)

elementData[i] = null;

size = 0;

}

　removeAll(collection c)：

1 public boolean removeAll(Collection<?> c) {

2 return batchRemove(c, false);//批量删除

3 }

batchRemove(xx,xx)：用于两个方法，一个removeAll()：它只清楚指定集合中的元素，retainAll()用来测试两个集合是否有交集。

//这个方法，用于两处地方，如果complement为false，则用于removeAll如果为true，则给retainAll()用，retainAll（）是用来检测两个集合是否有交集的。

private boolean batchRemove(Collection<?> c, boolean complement) {

final Object[] elementData = this.elementData; //将原集合，记名为A

int r = 0, w = 0; //r用来控制循环，w是记录有多少个交集

boolean modified = false;

try {

for (; r < size; r++)

//参数中的集合C一次检测集合A中的元素是否有，

if (c.contains(elementData[r]) == complement)

//有的话，就给集合A

elementData[w++] = elementData[r];

} finally {

// Preserve behavioral compatibility with AbstractCollection,

// even if c.contains() throws.

//如果contains方法使用过程报异常

if (r != size) {

//将剩下的元素都赋值给集合A，

System.arraycopy(elementData, r,

elementData, w,

size - r);

w += size - r;

}

if (w != size) {

//这里有两个用途，在removeAll()时，w一直为0，就直接跟clear一样，全是为null。

//retainAll()：没有一个交集返回true，有交集但不全交也返回true，而两个集合相等的时候，返回false，所以不能根据返回值来确认两个集合是否有交集，而是通过原集合的大小是否发生改变来判断，如果原集合中还有元素，则代表有交集，而元集合没有元素了，说明两个集合没有交集。

// clear to let GC do its work

for (int i = w; i < size; i++)

elementData[i] = null;

modCount += size - w;

size = w;

modified = true;

}

}

return modified;

}

4、总结

 　　　　这篇文章花费了我一上午的时间，真的很累，但是也很有收获，完全清楚了arrayList的实现，和它的一些重要的方法的原理

　　　　　　　　1、arrayList可以存放null。

　　　　　　　　2、arrayList本质上就是一个elementData数组。

　　　　　　　　3、arrayList区别于数组的地方在于能够自动扩展大小，其中关键的方法就是gorw()方法。

　　　　　　　　4、arrayList中removeAll(collection c)和clear()的区别就是removeAll可以删除批量指定的元素，而clear是全是删除集合中的元素。

　　　　　　　　5、arrayList由于本质是数组，所以它在数据的查询方面会很快，而在插入删除这些方面，性能下降很多，有移动很多数据才能达到应有的效果

　　　　　　　　6、arrayList实现了RandomAccess，所以在遍历它的时候推荐使用for循环。