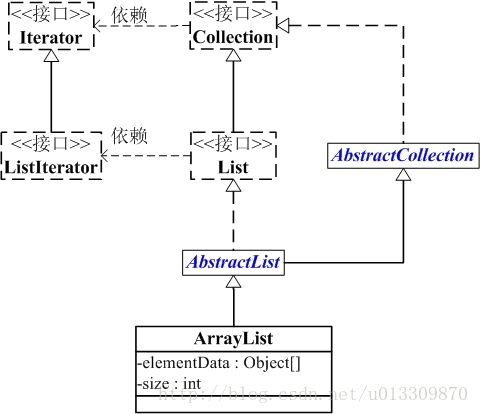
**ArrayList简介**

ArrayList就是动态数组，用MSDN(Microsoft Developer Network)中的说法，就是Array的复杂版本，它提供了动态的增加和减少元素，实现了Collection和List接口，可以灵活的设置数组的大小。要注意的是ArrayList并不是线程安全的，因此一般建议在单线程中使用ArrayList。

**ArrayList的继承关系**



public class ArrayList<E>

extends AbstractList<E>

implements List<E>,RandomAccess,Cloneable,Serializable

由上可知ArrayList继承AbstractList并且实现了List和RandomAccess，Cloneable,Serializable接口。

**ArrayList的方法使用和源码解析**

①构造方法（三种）

//1-----------------------

public ArrayList(){

this(10);

//调用ArrayList(10)默认初始化一个大小为10的object数组。

}

//2-------------------------

public ArrayList(int initialCapacity){

if(initialCapacity<0)

throw new IllegalArgumentException("Illegal Capacity:"+initialCapacity);

//如果用户初始化大小小于0抛异常，否则新建一个用户初始值大小的object数组。

this.elementData=new Object[initialCapacity];

}

//3--------------------------

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

elementData=c.toArray();

size=elementData.length;

//当c.toArray返回的不是object类型的数组时，进行下面转化。

if(elementData.getClass()!=Object[].class)

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

}

由上面三种构造方法可知，默认情况下使用ArrayList会生成一个大小为10的Object类型的数组。也可以调用ArrayList(int initialCapacity)来初始化Object数组的大小。并且用户可以往ArrayList中传入一个容器只要这个容器是Collection类型的。调用ArrayList(Collection<?extends E>c)接口的时候会将容器数组化处理并将这个数组值赋给Object数组。

实例：

public static void main(String[]args){

ArrayList<Integer>list\_2=new ArrayList<Integer>(20);

//list\_2中添加元素

for(int i=0;i<10;i++)

list\_2.add(i);

ArrayList<Integer>list\_3=new ArrayList<Integer>(list\_2);

//输出list\_2中元素

for(Integer a:list\_2)

System.out.print(a+"");

//输出list\_3中元素

for(Integer a:list\_3)

System.out.print(a+"");

}

//输出

/\*

list\_2:0 1 2 3 4 5 6 7 8 9

-----------------------

list\_3:0 1 2 3 4 5 6 7 8 9

\*/

②indexOf(Object o)方法

功能：查找某个元素在ArrayList中第一次出现的位置。

public int indexOf(Object o){

//ArrayList中的元素可以为null，如果为null返回null的下标

if(o==null){

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

if(elementData[i]==null) //elementData是一个缓存数组

return i;

}else{

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

if(o.equals(elementData[i]))

return i;

}

//如果没有找到对应的元素返回-1。

return-1;

}

对于indexof方法做几点说明：ArrayList中可以存放null元素，indexof是返回elementData数组中值相同的首个元素的下标，indexof中比较方法是equals而equals是比较元素的值，因此必须对null单独查找。如果未找到该元素则返回-1。

public static void main(String[]args){

ArrayList<Integer>list=new ArrayList<Integer>();

list.add(1);

list.add(2);

list.add(null);

list.add(2);

list.add(3);

System.out.println("null:"+list.indexOf(null));

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

System.out.println("2:"+list.indexOf(2));

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

System.out.println("4:"+list.indexOf(4));

}

//输出

/\*

null:2

-------------------------

2:1

-------------------------

4:-1

\*/

③lastIndexOf(Object o)方法

功能：查找某个元素在ArrayList中最后出现的位置。

public int lastIndexOf(Object o){

if(o==null){

//如果o为null从后往前找到第一个为null的下标

for(int i=size-1;i>=0;i--)

if(elementData[i]==null)

return i;

}else{

//从后往前找到第一个值为o的下标

for(int i=size-1;i>=0;i--)

if(o.equals(elementData[i]))

return i;

}

return-1;

}

上面代码做几点说明：lastIndexOf(Object o)在ArrayList中从后往前找到第一个跟要查找值相同的元素的下标，因为是按值查找所以对于null要单独查找。如果未找到则返回-1；

④get(int index)方法

功能：返回ArrayList中指定下标为index的元素。

public E get(int index){

//检查index的值是否大于ArrayList的大小

rangeCheck(index);

//返回index下标的元素

return elementData(index);

}

E elementData(int index){

return(E)elementData[index];

}

//这里值检查index>=size的情况，因为index<0时会自动抛出异常，所以并未检查index<0的情况。

private void rangeCheck(int index){

if(index>=size)

throw new IndexOutOfBoundsException(outOfBoundsMsg(index));

}

对上面代码做几点说明：上面代码中只检查了index>=size的情况，在index<0的情况下也会抛出异常，只是这个异常是由系统抛出的。index>=size要检查的原因是有可能数组的大小大于index，然而有效里面的元素<index这时不抛异常就会返回无效值。举个例子ArrayList的初始化大小为10，现在往里面放5个元素，如果index>=5时，应该要抛出异常，而不是返回null。因为null是可以主动放在ArrayList中的。

⑤set(int index,E element)方法

功能：将element放到ArrayList下标为index的位置，如果index<0或index>=size抛异常，set(int index,E element)只能覆盖ArrayList中原来的元素，返回值为被覆盖的元素。

public E set(int index,E element){

//检查index是否小于size，如果不是抛异常

rangeCheck(index);

E oldValue=elementData(index);

elementData[index]=element;

//覆盖ArrayList中index上的元素。

return oldValue;

//返回被覆盖的元素。

}

private void rangeCheck(int index){

if(index>=size)

throw new IndexOutOfBoundsException(outOfBoundsMsg(index));

}

⑥add(E e)方法

功能：往ArrayList中添加元素。

//1-----------------------

public boolean add(E e){

ensureCapacityInternal(size+1);//加入元素前检查数组的容量是否足够

elementData[size++]=e;

return true;

}

//2-----------------------

private void ensureCapacityInternal(int minCapacity){

modCount++;

//如果添加元素后大于当前数组的长度，则进行扩容

if(minCapacity-elementData.length>0)

grow(minCapacity);

}

//3-----------------------

private void grow(int minCapacity){

//overflow-conscious code

int oldCapacity=elementData.length;

/\*将数组的长度增加原来数组的一半。

>>位运算，右移一位。整体相当于

newCapacity = oldCapacity + 0.5 \* oldCapacity

jdk1.7采用位运算比以前的计算方式更快

\*/

int newCapacity=oldCapacity+(oldCapacity>>1);

if(newCapacity-minCapacity<0)

newCapacity=minCapacity;

//如果扩充一半后仍然不够，则newCapacity=minCapacity;minCapacity实际元素的个数。

if(newCapacity-MAX\_ARRAY\_SIZE>0)

newCapacity=hugeCapacity(minCapacity);

//数组最大位2^32

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

elementData=Arrays.copyOf(elementData,newCapacity);

}

add方法比较复杂，涉及到扩充数组容量的问题。其中要弄清楚size和elementData.length的区别，size指的是数组中存放元素的个数，elementData.length表示数组的长度，当new一个ArrayList系统默认产生一个长度为10的elementData数组，elementData.length=10，但是由于elementData中还未放任何元素所有size=0。如果加入元素后数组大小不够会先进行扩容，每次扩容都将数组大小增大一半比如数组大小为10一次扩容后的大小为10+5=15;ArrayList的最大长度为2^32.

⑦add(int index,E element)方法

功能：往ArrayList指定index上添加元素，添加元素后ArrayList的大小增1。index及以后的元素都会向后移一位。

//1-------------------------

public void add(int index,E element){

rangeCheckForAdd(index);

//检查index的值是否在0到size之间，可以为size。

ensureCapacityInternal(size+1);//看elementData的长度是否足够，不够扩容

//将elementData从index开始后面的元素往后移一位。

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

size-index);

elementData[index]=element;

size++;

}

//2-------------------------

private void rangeCheckForAdd(int index){

if(index>size||index<0)

throw new IndexOutOfBoundsException(outOfBoundsMsg(index));

}

//3-------------------------

private void ensureCapacityInternal(int minCapacity){

modCount++;

//overflow-conscious code

if(minCapacity-elementData.length>0)

grow(minCapacity);

}

add(int index,E element)往指定index中加入元素，加入元素之前先检查数组的大小，如果小了在原来基础上增大一半，将elementData从index开始后面的元素往后移一位，将element放到index位置。

⑧remove(int index)方法

功能：删除ArrayList指定位置的元素。

public E remove(int index){

rangeCheck(index);

//如果index>=size抛出异常

modCount++;

E oldValue=elementData(index);

//获取删除元素的值

int numMoved=size-index-1;

//将index后面所有的元素往前移一位。

if(numMoved>0)

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

numMoved);

elementData[--size]=null;//Let gc do its work

//返回要删除的原数。

return oldValue;

}

⑨remove(Object o)方法

功能：删除ArrayList中值为o的元素

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;

}