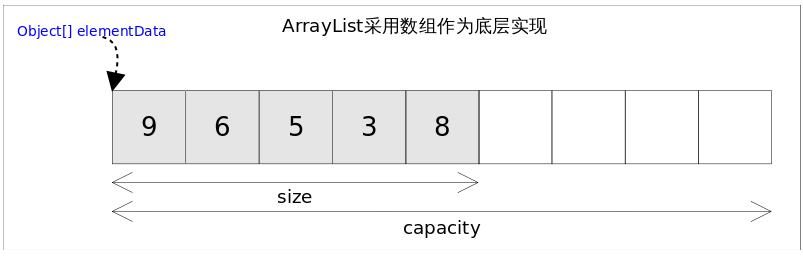
1. ArrayList概念相关
2. 基本介绍
3. ArrayList实现了List接口，是顺序容器，即元素存放的数据与放进去的顺序相同，允许放入null元素，底层通过大小可变的数组（Resizable Array）实现。
4. ArrayList没有实现同步，就是非线程安全，其余跟Vector大致相同。
5. 每个ArrayList都有一个容量（capacity），表示底层数组的实际大小，容器内存储元素的个数不能多于当前容量。当向容器中添加元素时，如果容量不足，容器会自动增大底层数组的大小。前面已经提过，Java泛型只是编译器提供的语法糖，所以这里的数组是一个Object数组，以便能够容纳任何类型的对象。
6. 底层实现



1. 注意事项
2. size(), isEmpty(), get(), set()方法均能在常数时间内完成。
3. add()方法的时间开销跟插入位置有关。
4. addAll()方法的时间开销跟添加元素的个数成正比。其余方法大都是线性时间。
5. 为追求效率，ArrayList没有实现同步（synchronized），如果需要多个线程并发访问，用户可以手动同步，也可使用Vector替代
6. ArrayList容量详解
   1. 初始容量和容量变化
7. List arrayList = new ArrayList();

像上面这样使用默认的构造方法，初始容量被设置为10。当ArrayList中的元素超过10个以后，会重新分配内存空间，使数组的大小增长到16。

1. List arrayList = new ArrayList(4);

这种声明方式，初始容量被设置为4。当超过4个后，会重新分配数组空间。

1. 容量变化的规则：((旧容量 \* 3) / 2) + 1
2. 一旦容量发生变化，就要带来额外的内存开销，和时间上的开销。
3. 数组最大容量为：private static final int MAX\_ARRAY\_SIZE = 2147483639;
   1. ArrayList自动改变数组长度的原理
4. ArrayList类的实质

ArrayList底层采用Object类型的数组实现，当使用不带参数的构造方法生成ArrayList对象时，实际上会在底层生成一个长度为10的Object类型数组。

1. 首先，ArrayList定义了一个私有的未被序列化的数组elementData，用来存储ArrayList的对象列表（注意只定义未初始）：

private transient Object[] elementData;

1. 其次，以指定初始容量（Capacity）或把指定的Collection转换为引用型数组后实例化elementData数组；如果没有指定，则预置初始容量为10进行实例化。把私有数组预先实例化，然后通过copyOf方法覆盖原数组，是实现自动改变ArrayList的大小(size)的关键。
2. 构造方法1:

public ArrayList(int paramInt)

{

if (paramInt > 0) {

//初始化为Object数组

elementData = new Object[paramInt];

} else if (paramInt == 0) {

//参数为0,初始化为空Object数组

elementData = EMPTY\_ELEMENTDATA;

} else {

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

}

}// 当声明数组为2147483630左右时，会发生内存溢出。OutOfMemoryError

1. 构造方法2：

public ArrayList()

{

//当不传参数时，初始化为长度为0 的Object数组

elementData = DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA;

}

1. 构造方法3：（传入参数为Collection类）

public ArrayList(Collection<? extends E> paramCollection)

{

//用Collection类初始化elementData数组

elementData = paramCollection.toArray();

if ((size = elementData.length) != 0)

{

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

//用copyOf方法覆盖原数组实现数组长度增长

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

}

}

else {

elementData = EMPTY\_ELEMENTDATA;

}

}

1. ArrayList自动改变size的机制

为了实现这一机制，java引进了Capacity和size概念，以区别数组的length。为了保证用户增加新的列表对象，java设置了最小容量（minCapacity），通常情况上，它大于列表对象的数目，所以Capactiy虽然就是底层数组的长度（length），但是对于最终用户来讲，它是无意义的。而size存储着列表对象的数量，才是最终用户所需要的。为了防止用户错误修改，这一属性被设置为private的，不过可以通过size()获取。

1. 初始的Capacity和size的值

Capacity初始值（initialCapacity）可以由用户直接指定或由用户指定的Collection集合存

储的对象数目确定。而size的被声明为int型变量，默认为0，当用户指定Collection创建ArrayList时，size值等于initialCapacity。

1. add()方法 - 改变数组长度的思路
2. public boolean add(E paramE) -- 添加一个对象

{

//size增加1，调整底层数组长度

ensureCapacityInternal(size + 1);

//将新添加的对象，增加到数组的size+1的位置

elementData[(size++)] = paramE;

return true;

}

1. private void ensureCapacityInternal(int paramInt)

{ //elementData为底层数组，paramInt为size+1

ensureExplicitCapacity(calculateCapacity(elementData, paramInt));

}

1. private static int calculateCapacity(Object[] paramArrayOfObject, int paramInt)

{ //当数组为空数组时，返回默认10或者 (size+1大于10)

if (paramArrayOfObject == DEFAULTCAPACITY\_EMPTY\_ELEMENTDATA) {

return Math.max(10, paramInt);

}

return paramInt;

}

1. private void ensureExplicitCapacity(int paramInt)

{ //modCount表示修改次数

modCount += 1;

//当ArrayList所需长度大于elementData的实际长度时进行长度扩充

if (paramInt - elementData.length > 0) {

grow(paramInt);

}

}

1. private void grow(int paramInt)

{ // i为目前的长度，j为增加后的长度

int i = elementData.length;

int j = i + (i >> 1);

//当j小于所需size时，j赋值为所需size

if (j - paramInt < 0) {

j = paramInt;

}

//当j大于默认最大数组长度时，扩充为integer.MAX\_VALUE

if (j - 2147483639 > 0) {

j = hugeCapacity(paramInt);

}

//利用copyOf方法，将数组长度扩充为j覆盖原来数组

elementData = Arrays.copyOf(elementData, j);

}

1. private static int hugeCapacity(int paramInt)

{

if (paramInt < 0) {

throw new OutOfMemoryError();

}

//当所需长度大于最大长度时，赋值为integer.MAX\_VALUE

return paramInt > 2147483639 ? Integer.MAX\_VALUE : 2147483639;

}

* 自动增加ArrayList数组大小的思路：向ArrayList添加对象时，原对象数目加1如果大于原底层数组长度，则以适当长度新建一个原数组的拷贝，并修改原数组，指向这个新建数组。原数组自动抛弃（java垃圾回收机制会自动回收）。size则在向数组添加对象，自增1。

1. remove()方法 - 改变数组长度
2. public E remove(int paramInt) - 删除一个对象By index

{ //当删除的index大于等于size时会报错

rangeCheck(paramInt);

modCount += 1;

Object localObject = elementData(paramInt);

// i表示 paramInt到最后一个对象的长度

int i = size - paramInt - 1;

if (i > 0) {

//将paramInt后的对象子数组移动到paramInt开始位置，相当于向左移动一位

System.arraycopy(elementData, paramInt + 1, elementData, paramInt, i);}

//size自减一位，数组最后一个对象置为空，会由java回收null

elementData[(--size)] = null;

//返回被删除的对象

return (E)localObject;

}

1. 变长的过程

综上所述，在用户向ArrayList追加对象时，Java总是要先计算容量（Capacity）是否适当，若容量不足则把原数组拷贝到以指定容量为长度创建的新数组内，并对原数组变量重新赋值，指向新数组。在这同时，size进行自增1。在删除对象时，先使用拷贝方法把指定index后面的对象前移1位（如果有的话），然后把空出来的位置置null，交给Java收集器销毁，size自减1，即完成了。

* 1. ArrayList数组扩容的核心方法

1. java.lang.System

public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)

将指定源数组中的数组从指定位置复制到目标数组的指定位置。 阵列组件的一个子序列被从通过引用的源阵列复制src被引用的目标阵列dest 。 复制的组件数量等于length参数。 源阵列中位置srcPos至srcPos+length-1的组件分别复制到目标阵列的位置destPos至destPos+length-1 。

1. 会抛出的异常
2. IndexOutOfBoundsException - 如果复制将导致数据外部数据界限的访问。
3. ArrayStoreException - 如果由于类型不匹配， src数组中的元素无法存储到 dest数组中。
4. NullPointerException - 如果 src或 dest是 null 。
5. ArrayList具体使用实例
6. 添加元素
   1. add() 方法
7. public boolean add(E paramE)

将指定的元素追加到此列表的末尾。

1. public void add(int paramInt, E paramE) -- 会抛异常IndexOutOfBoundsException

* 先把数组往后移动一位，然后利用数组elementData[index]赋值。

在此列表中的指定位置插入指定的元素。 将当前位于该位置的元素（如果有）和任何后续元素（向其索引添加一个）向右移动一位。

* 1. set() 方法

1. public E set(int paramInt, E paramE) -- 会抛异常IndexOutOfBoundsException

用指定的元素替换此列表中指定位置的元素。

* 1. addAll() 方法

1. public boolean addAll(Collection<? extends E> paramCollection)

按指定集合的Iterator返回的顺序将指定集合中的所有元素追加到此列表的末尾。 如果在操作进行中修改了指定的集合，则此操作的行为是未定义的。 （这意味着如果指定的集合是此列表，则此调用的行为是未定义的，并且此列表是非空的。）

抛出的异常：NullPointerException - 如果指定的集合为空

1. public boolean addAll(int paramInt, Collection<? extends E> paramCollection)

将指定集合中的所有元素插入到此列表中，从指定的位置开始。 将当前位于该位置（如果有的话）的元素和随后的任何元素移动到右边（增加其索引）。 新元素将按照指定集合的迭代器返回的顺序显示在列表中。

原理： 先将插入位置后面的元素向右移动和要插入数组长度一样的位数，相当于空出了插入数组长度的位置，再将要插入数组复制到这段位置，插入完成。

异常： IndexOutOfBoundsException - 如果指数超出范围（ index < 0 || index > size() ）

NullPointerException - 如果指定的集合为空

1. 获取元素
   1. get() 方法
2. public E get(int paramInt)

{

rangeCheck(paramInt); // 边界检测

return (E)elementData(paramInt); // 取出元素进行类型转换

}

底层数组是Object[],取出元素后要进行类型转换。

* 1. 迭代器方法取出元素（后面再说）

1. 删除元素

3.1 remove() 方法

(1) public E remove(int paramInt)

原理：先将数组往左移动一位，利用数组赋值System.arraycopy方法实现，再把最后一位置为空。

说明：删除该列表中指定位置的元素。 将任何后续元素移动到左侧,向左移动一位。（从其索引中减去一个元素）。

1. public boolean remove(Object paramObject)

{

int i;

// 如果删除的元素为空

if (paramObject == null) {

//循环全部元素，删除第一个为空的元素

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

if (elementData[i] == null)

{ //删除指定index的元素，并将后面的元素往前移动一位

fastRemove(i);

return true;

}

} //如果要删除的元素不为空

} else {

for (i = 0; i < size; i++) { //循环判断相等的元素，然后删除

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

{

fastRemove(i);

return true;

}

}

}

return false;

} //删除成功则返回true，如果数组中没有指定元素则返回false

**从列表中删除指定元素的第一个出现（如果存在）**。 如果列表不包含该元素，则它不会更改。 更正式地，删除具有最低索引i的元素， 如果此列表包含指定的元素（或等效地，如果此列表作为调用的结果而更改），则返回true 。

1. protected void removeRange(int paramInt1, int paramInt2)

从此列表中删除所有索引为fromIndex （含）和toIndex之间的元素。 将任何后续元素移动到左侧（减少其索引）。 此通话由(toIndex - fromIndex)元素缩短列表。 （如果是toIndex==fromIndex ，这个操作没有效果）

源码：protected void removeRange(int paramInt1, int paramInt2)

{

modCount += 1;

//将toIndex后面的元素移动到fromIndex开始的位置

int i = size - paramInt2;

System.arraycopy(elementData, paramInt2, elementData, paramInt1, i);

//将多余的位置元素置为空，方便GC回收

int j = size - (paramInt2 - paramInt1);

for (int k = j; k < size; k++) {

elementData[k] = null;

}

size = j;

}

3.2 clear()方法

public void clear()

{

modCount += 1;

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

elementData[i] = null;

}

size = 0;

}

循环赋值为null，等待垃圾处理器回收

3.3 内存泄漏的说明

有了垃圾收集器并不意味着一定不会有内存泄漏。**对象能否被GC的依据是是否还有引用指向它**，上面代码中如果不手动赋null值，除非对应的位置被其他元素覆盖，否则原来的对象就一直不会被回收。所以删除元素，必须将最后一位赋值为null。

1. 其他方法
   1. 复制数组
2. public Object clone() -- 复制新的ArrayList对象，返回一个Object，方法Arrays.copyOf。
3. 修改复制的ArrayList对象，不会影响到旧的ArrayList对象。
4. Arrays.copyOf底层也是调用了System.arraycopy方法。
5. toArray ()
6. public Object[] toArray() //复制一个新数组Object[]，复制值而不是复制引用

{

return Arrays.copyOf(elementData, size);

}

说明： 以正确的顺序（从第一个到最后一个元素）返回一个包含此列表中所有元素的数组。

返回的数组将是“安全的”，因为该列表不保留对它的引用。 （换句话说，这个方法必须分配一个新的数组）。 因此，调用者可以自由地修改返回的数组。

1. public <T> T[] toArray(T[] paramArrayOfT) //将ArrayList复制到指定数组

{ //当目标数组的长度小于操作list时，目标数组为null

if (paramArrayOfT.length < size) {

return (Object[])Arrays.copyOf(elementData, size, paramArrayOfT.getClass());

}

System.arraycopy(elementData, 0, paramArrayOfT, 0, size);

//当目标数组长度大于操作list时，全部复制过去并且将目标数组的size位设为空。其他值也为空。

if (paramArrayOfT.length > size) {

paramArrayOfT[size] = null;

}

return paramArrayOfT;

}

* 1. 判断是否存在相同元素

public boolean contains(Object o) -- 如果有存在至少一个相同元素，返回true。

* 1. 返回指定元素的索引

public int indexOf(Object paramObject) -- 返回相同元素的最小索引

原理：先判断是否为空元素，再利用equals方法判断是否相同，只会返回最小的那个索引。

如果不含有这个元素，则返回 -1.

4.4 判断list是否为空

public boolean isEmpty()

原理：如果size==0则返回空，就是说如果存在空值，只要size>1,那就不为空。

* 1. 截取一个子List

public List<E> subList(int paramInt1, int paramInt2)

subList是一个新class，继承ArrayList。

返回指定的fromIndex （含）和toIndex之间的列表部分的视图。

1. 对原来的list和返回的list做的“非结构性修改”，都会影响到彼此对方。就是对list的修改会影响到subList，对subList修改会影响到list。

所谓的“非结构性修改”，是指不涉及到list的大小改变的修改。相反，结构性修改，指改变了list大小的修改。

1. 如果对list进行结构性修改，subList会抛出异常ConcurrentModificationException。

所以对list进行结构性修改后，应该重新返回一个subList。

* 1. 对ArrayList进行排序

1. 使用Collections.sort()传入ArrayList，会采用默认的方式进行排序（字典序）
2. 使用Collections.sort()传入ArrayList和自己实现Commparator接口的类的对象，实现自定义排序
3. 使用List.sort()传入自己实现Commparator接口的类的对象，实现自定义排序

public void sort(Comparator<? super E> paramComparator)

1. 总结
2. ArrayList的特点。 -- 顺序存储，数据可重复,内部采用动态数组实现
3. 可以随机访问，按照索引位置进行访问效率很高，用算法描述中的术语，效率是O(1)，简单说就是可以一步到位。
4. 除非数组已排序，否则按照内容查找元素效率比较低，具体是O(N)，N为数组内容长度，也就是说，性能与数组长度成正比。
5. 添加元素的效率还可以，重新分配和拷贝数组的开销被平摊了，具体来说，添加N个元素的效率为O(N)。
6. 插入和删除元素的效率比较低，因为需要移动元素，具体为O(N)。