链滴

ArrayList

作者：helly
原文链接：https://ld246.com/article/1510288048074
来源网站：链滴

java.util.ArrayList

允许为空；

允许重复数据；

有序；

非线程安全。

```
public class ArrayList<E> extends AbstractList<E> implememts List<E>, RondomAccess, Clonable, java.io.Serializable {
  private static final int DEFAULT_CAPACITY = 10;

  // 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elementDta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，而不是整个组，从而节省空间和时间。
  transient Object[] elementData; // 非私有简化嵌套类访问

  private int size; // 逻辑长度

  public  ArrayList() {
    elementData = DEFAULTCAPACITY_EMPTY_ELEMENTDATA; // 为什么这里选择空数组而不是始化为容量为10的数组呢？其实是用了懒加载，第一次添加元素时会自动扩容为容量10
  }

  // 先把集合转化为数组，然后判断数组类型是否是Object[].class（如果不是的话，不属于此类及其类但属于Object类或其子类的实例就添加不进去了），不是的话通过Arrays.copyOf()复制（copyOf层靠System.copyArray()方法）
  public ArrayList(Collection<? extens E> c) {
    elementData = c.toArray(); // 用Arrays.copyOf()实现
    if ((size = elementData.length) != 0) {
      // c.toArray might (incorrectly) not return Object[] (see 6260652)
      if (elementData.getClass() != Object[].class)
        elementData = Arrays.copyOf(elementData, size, Object[].class);
    } else {
      array.  this.elementData = EMPTY_ELEMENTDATA;
    }
  }

  public Object[] toArray() {
    return Arrays.copyOf(elementData, size);
  }

  /*
  * 先判断是否是第一次添加元素，是的话懒加载默认容量10，添加元素
  * 否则断数组长度是否大于逻辑长度，是的话直接添加元素，不是的话进行数组扩容；
  * 将数组长度扩大为1.5倍，然后进一步判断；
  * 如果数组长度依然比请求的最小长度小，将数组直接扩容到请求的长度（这种情况发生在：一次性入多个值，也就是调用addAll()方法）
  * 如果数组长度大于2\^31-9且小于等于2\^31-1（因为有一个符号位），返回2\^31-1。（2^31-是因为size是int类型）
```

原文链接：ArrayList

```
 */
 public boolean add(E e) {
   ensureCapacityInternal(size + 1);
   elementData[size++] = e;
   return true;
 }

 private void ensureCapacityInternal(int minCapacity) {
   // 第一次添加元素懒加载默认容量10
   if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
     minCapacity = Math.max(DEFAULT_CAPACITY, minCapacity);
   }

   ensureExplicitCapacity(minCapacity);
 }

 private void ensureExplicitCapacity(int minCapacity) {
   modCount++; // 记录修改次数，modCount在AbstractList类中声明protected transient int m
dCount = 0;

 // 数组越界，需要进行容量扩展
 if (minCapacity - elementData.length > 0)
 grow(minCapacity);
 }

 /**
 * The maximum size of array to allocate.
 * Some VMs reserve some header words in an array.
 * Attempts to allocate larger arrays may result in
 * OutOfMemoryError: Requested array size exceeds VM limit
 * -8 是为了减少出错的几率
 */
 private static final int MAX_ARRAY_SIZE = Integer.MAX_VALUE - 8;

 private void grow(int minCapacity) {
   int oldCapacity = elementData.length;
   int newCapacity = oldCapacity + (oldCapacity >> 1); // 相当于oldCapacity + (oldCapacity /
2)

   if (newCapacity - minCapacity < 0)
     newCapacity = minCapacity;

   if (newCapacity - MAX_ARRAY_SIZE > 0)
     newCapacity = hugeCapacity(minCapacity);

   elementData = Arrays.copyOf(elementData, newCapacity);
 }

 private static int hugeCapacity(int minCapacity) {
   if (minCapacity < 0) // overflow，因为0111 .... 1111 + x 只要 x 大于 1 就进位，符号位变为 1
为负数
     throw new OutOfMemoryError();

   return (minCapacity > MAX_ARRAY_SIZE) ? Integer.MAX_VALUE : MAX_ARRAY_SIZE;
```

```
}

public  void add(int  index, E element)

public  boolean  addAll(Collection c)

public E get(int  index)

public  int  indexOf(Object o)

public  int  lastIndexOf(Object o)

public E remove(int  index)

// 根据删除的对象是否为null进行区分，因为是null就不能调用equals方法了
public  boolean remove(Object o)

private  void fastRemove(int  index)

// 要求集合c不能为null
public  boolean  removeAll(Collection c)

public  void  clear()

public Iterator<E> iterator() {
    return new Itr();
}

private class Itr implements Iterator<E> {
    int cursor;       // index of next element to return
    int lastRet = -1; // index of last element returned; -1 if no such
    int expectedModCount = modCount;

    public boolean hasNext() {
        return cursor != size;
    }

    @SuppressWarnings("unchecked")
    public E next() {
        checkForComodification();
        int i = cursor;
        if (i >= size)
            throw new NoSuchElementException();
        Object[] elementData = ArrayList.this.elementData;
        if (i >= elementData.length)
            throw new ConcurrentModificationException();
        cursor = i + 1;
        return (E) elementData[lastRet = i];
    }

   //检查modCount是否改变了，如果改变了，直接抛出异常
   final  void checkForComodification() {
     if (modCount != expectedModCount)
       throw  new ConcurrentModificationException();
```

```
    }
  }

  // 双向移动
  public ListIterator listIterator(int  index)

  private  class ListItr extends Itr implements ListIterator<E>
```

Verctor：

1、线程安全；

2、Vector可以指定增长因子，如果该增长因子指定了，那么扩容的时候会每次新的数组大小会在原组的大小基础上加上增长因子；如果不指定增长因子，那么就给原数组大小*2。

```
int newCapacity = oldCapacity + ((capacityIncrement >  0  ) ? capacityIncrement : oldCapacity)
```

数组和List相互转化：

```
List<String> strList = Arrays.asList(arr);
String[] arr = strList.toArray();
```

类.class

对象.getClass()

参考：

ArrayList源码

http://blog.csdn.net/qq_19431333/article/details/54405686

http://blog.csdn.net/shijinupc/article/details/7827507