List（很多list都是继承他的）

1. Collection.reverse()，翻转数组（翻转的是ArrayList，而普通的int[]数组是不能用reverse翻转）
2. List.toArray(); //把动态数组转化为数组，这一个返回是副本，原先引用不会改变
3. toArray()返回的是泛型 Object[]，
4. 想对返回的Object进行强转：list.toArray(new int[0][]);
5. 传参一个目标数组的对象类型 如new String[0] ， 长度设为0
6. 泛型必须是引用数据类型，但是int[]一维数组并不是引用数据类型，可是二维数组int[][]就是引用数据类型

Arraylist<> 动态数组，可以储存泛型

1. Add(e) , add(index , e)添加至数组尾部
2. Remove(E e)删除 （arraylist.remove(index)用下标来删除！！！！！）
3. Get() , get(index)访问列表
4. Iterator iter = arraylist.iterator()
   1. Iter.hasNext();
   2. Iter.next();

Linkedlist<> 是链表结构，访问起来或去中间元素是效率很低的

1. linkedlist和arraylist有一样的方法，但是算法的内部不一样，效率不一样，不同需求选不同的list
2. getFirst()
3. getLast() // linkedlist访问头尾节点是速度很快的
4. linkedlist也就是队列：
   1. 入队：add
   2. 出对: remove(0) //要输入参数！！！！ （相当于 pop()的效果，还能返回队头元素，linkedlist不能用pop）

linkedlist也就是队列

public void clear()，清空链表；

public E removeFirst()，删除并返回第一个元素；

public E removeLast()，删除并返回最后一个元素；

public boolean remove(Object o)，删除某一元素，返回是否成功；

public E remove(int index)，删除指定位置的元素；

public E poll()，删除并返回第一个元素；

**public E remove()，删除并返回第一个元素；**



Arrays.sort(List); // 默认升序

Arrays.sort(List , new Comparator<>(){ //正升序反降序

@Override

Public int compare(Integer a , Integer b){

Return b - a;

}

})

Arrays.sort(List , start\_index , end\_index); // 排序下标范围是前闭后开

关于ArrayList和LinkedList

1. Arraylist.add(Object e);
2. Arraylist.add(int index , Object e); //在index的位置插入元素
3. 将Arraylist转化为数组
   1. Object[] res = arraylist.toArray(); //注意这里输出的是一个Object[]的数组
   2. 假设有 int[] res = arraylist.toArray(); //这个代码是会报错的，原因是右边toArray返回的是一个Object[]的数组，将一个Object的数组赋给一个int[]数组，这属于向下转换，是要强制转换的（向上才可以直接转换）
   3. 如何解决，将toArray()时返回的不是Object而是int，在括号中加上泛型

Int[] res = arraylist.toArray(new int[1]);

* 1. public Object[] toArray();
  2. public <T> T[] toArray(T[] a);