* 学习目标

  * 能够实现自定义ArrayList

  * 能够说出ArrayList和LinkedList区别

---

  * 能够实现自定义ArrayList

```
* 需要实现功能
 * 添加元素：
    * 默认添加
     public boolean add(E e);
    * 根据位置添加
     public void add(int index, E element)
 * 根据索引删除某个元素
    * public E remove(int index)
 * 根据索引获取某个元素
    * public E get(int index)
 * 获取列表大小：
    * public int size()
 * 判断列表是否为空
    * public boolean isEmpty()
 * 清除列表
    * public void clear();

* 实现思路
  * 构建ArrayList
   * 定义常量：默认容量的大小
    * DEFAULT_CAPACITY=10
   * 定义常量：数组的最大值
    * MAX_ARRAY_SIZE=Integer.MAX_VALUE-8
   * 定义常量：空数组
    * Object[] DEFAULTCAPACITY_EMPTY_ELEMENTDATA= {};
   * 定义数组
    * Object[] elementData
   * 数组的大小
```

```
 * size
* 定义无参数构造器
  * 数组赋值为空数组（在添加的时候，才初始化容易）
     * public ArrayList() {
         elementData=DEFAULTCAPACITY_EMPTY_ELEMENTDATA;
         }
* 定义有参数的构造（参数为初始化容量）
  * 判断initialCapacity
  * 假如大于0：直接根据初始化容量构建内部数组elementData
  * 假如等于0：用默认容量构建内部数组elementData
  * 假如小于0：直接抛出异常
  public ArrayList(int initialCapacity) {
     if(initialCapacity>0) {
         elementData=new Object[initialCapacity];
     }else if(initialCapacity==0) {
         elementData=new Object[DEFAULT_CAPACITY];
     }else {
         throw new IllegalArgumentException("Illegal capacity:"+initialCapac
     }
  }
* 获取列表大小
  * return size
* 判断列表是否为空
  * return size==0
* 添加元素:public boolean add(E e);
   * 确保内部数组容量，可以添加元素 ensureCapacityInternal
     * 计算容量（主要看数组是否为空，为空需要初始化）calculateCapactiy
      * 判断当前数组是空数组，假如空数组的：取默认初始化容量和minCapacity最大值
      *假如不为空直接返回minCapacity
     * 确保可以扩展内部数组容量ensureExplicitCapacity
       * 判断minCapacity是否已经超过内部数组的大小
        * 假如超过了就扩展 if(minCapacity-elementData.length>0)
        * 扩展数组的容量 grow
          * 保存之前老数组的长度
          * 扩展之前容量的一半
          * 假如扩展之后还是比minCapacity小，就用minCapacity
          * 假如扩展之后比较MAX_ARRAY_SIZE大if(newCapacity-MAX_ARRAY_SIZE>0)
             * 需要通过minCapacity重新就是一个新的大小 hugeCapacity
                * 假如minCapacity变成负数就抛OutOfMemoryError
                * 假如minCapacity大于MAX_ARRAY_SIZE返回Integer.MAX_VALUE否则
```

```
69              * 创建新的数组：copy之前值，长度为newCapacity
70      * 给数组的下一个坐标赋值
71          elementData[size++]=e;
72      * 返回true
73   * 根据位置添加 void add(int index, E element)
74     * 检查添加的下标范围是否合法 rangeCheckForAdd
75          if(index<0 || index>size) {
76              outOfBoundsMsg(index);
77          }
78         private void outOfBoundsMsg(int index) {
79          throw new IndexOutOfBoundsException("Size:"+size+",Index:"+index);
80          }
81     * 确保内部数组容量，可以添加元素
82          ensureCapacityInternal
83     * 通过复制数组形式：腾出位置(建议画图理解)
84          System.arraycopy(elementData, index, elementData, index+1, size-index);
85     * 当前数组下标位置赋值
86          elementData[index]=e;
87     * size 加一
88        size++
89   * 根据索引获取某个元素 E get(int index)
90      * 下标范围检测 checkRange
91          if(index>=size) {
92              outOfBoundsMsg(index);
93          }
94      * 返回下标数组的值
95        (E) elementData[index]
96  * 根据索引删除某个元素: E remove(int index)
97    * 检测下标是否在合理范围
98      checkRange(index);
99    * 获得要移除元素
100     E oldValue=(E) elementData[index];
101   * 通过通数组copy：移除该元素
102     int numMoved=size-index-1;
103        if(numMoved>0) {
104             System.arraycopy(elementData, index+1, elementData, index, numMoved
105         }
106   * 释放最后一个资源
107     elementData[size--]=null;
108   * 返回删除的值
```

```
     return oldValue;

 * 清除列表 void clear()
  * 所有元素置为空
    for (int i = 0; i < size; i++) {
            elementData[i]=null;
        }
  * size 设置为0
   size=0;

*  代码
import java.util.Arrays;

public class ArrayList<E>{
    /**
     * 默认容量的大小
     */
    private static final int DEFAULT_CAPACITY=10;

    /**
     * ArrayList 数组最大值
     */
    private static final int MAX_ARRAY_SIZE=Integer.MAX_VALUE-8;
    /**
     * 空数组
     */
    private static final Object[] DEFAULTCAPACITY_EMPTY_ELEMENTDATA= {};
    /**
     * ArrayList 底层使用的数组
     * 任何空的数组（elementData==DEFAULTCAPACITY_EMPTY_ELEMENTDATA），在第一次被
     * ，容量都被赋值为DEFAULT_CAPACITY
     */
    private Object[] elementData;

    /**
     * ArrayList的大小
     */
    private int size;

    /**
```

```
     * 默认初始10个，在添加的时候再初始化，先给空数组
     */
    public ArrayList() {
        elementData=DEFAULTCAPACITY_EMPTY_ELEMENTDATA;
    }

    /**
     * @param initialCapacity：自定义初始化容量
     */
    public ArrayList(int initialCapacity) {
        // 判断initialCapacity
        // 假如大于0：直接根据初始化容量构建内部数组elementData
        // 假如等于0：用默认容量构建内部数组elementData
        // 假如小于0：直接抛出异常
        if(initialCapacity>0) {
            // 假如大于0：直接根据初始化容量构建内部数组elementData
            elementData=new Object[initialCapacity];
        }else if(initialCapacity==0) {
            // 假如等于0：用默认容量构建内部数组elementData
            elementData=new Object[DEFAULT_CAPACITY];
        }else {
            // 假如小于：直接抛出异常
            throw new IllegalArgumentException("Illegal capacity:"+initialCapac
        }
    }
    public boolean add(E e) {
        // 确保内部数组容量，可以添加元素
        ensureCapacityInternal(size+1);
        //给数组的下一个坐标赋值
        elementData[size++]=e;
        return true;
    }

    public void add(int index,E e) {
        // 检查添加的下标范围是否合法
        rangeCheckForAdd(index);
        // 确保内部数组容量，可以添加元素
        ensureCapacityInternal(size+1);
        // 通过复制数组形式：腾出位置(建议画图理解)
        System.arraycopy(elementData, index, elementData, index+1, size-index);
```

```
        // 当前数组下标位置赋值
        elementData[index]=e;
        // size 加一
        size++;
    }
    @SuppressWarnings("unchecked")
    public E get(int index) {
        // 下标范围检测
        checkRange(index);
        // 返回下标数组的值
        return (E) elementData[index];
    }

    /**
     * @param index
     * 通过下标移除，返回当前被删除的元素
     */
    @SuppressWarnings("unchecked")
    public E remove(int index) {
        // 检测下标是否在合理范围
        checkRange(index);
        // 获得要移除元素
        E oldValue=(E) elementData[index];
        // 通过通数组copy：移除该元素
        int numMoved=size-index-1;
        if(numMoved>0) {
            System.arraycopy(elementData, index+1, elementData, index, numMoved
        }
        // 释放最后一个资源
        elementData[--size]=null;// 让垃圾回收资源
        // 返回删除的值
        return oldValue;
    }

    /**
     * 清空列表
     */
    public void clear() {
        // 所有元素置为空
        for (int i = 0; i < size; i++) {
```

```
229             elementData[i]=null;
230         }
231         // size 设置为0
232         size=0;
233     }
234 
235     private void checkRange(int index) {
236         if(index>=size) {
237             outOfBoundsMsg(index);
238         }
239     }
240 
241     /**
242      * @param index
243      * 检查添加的下标范围是否合法
244      */
245     private void rangeCheckForAdd(int index) {
246         if(index<0 || index>size) {
247             outOfBoundsMsg(index);
248         }
249     }
250 
251     private void outOfBoundsMsg(int index) {
252         throw new IndexOutOfBoundsException("Size:"+size+",Index:"+index);
253     }
254 
255     private void ensureCapacityInternal(int minCapacity) {
256         // 计算容量（主要看数组是否为空，为空需要初始化）
257         minCapacity=calculateCapactiy(elementData,minCapacity);
258         // 确保可以扩展内部数组容量
259         ensureExplicitCapacity(minCapacity);
260     }
261 
262     /**
263      * @param minCapacity
264      * 确保可以扩展内部数组容量
265      */
266     private void ensureExplicitCapacity(int minCapacity) {
267         // 判断minCapacity是否已经超过内部数组的大小
268         // 假如超过了就扩展
```

```
269         if(minCapacity-elementData.length>0) {
270             //扩展数组的容量
271             grow(minCapacity);
272         }
273     }
274 
275     /**
276      * @param minCapacity
277      * 扩展数组的容量
278      */
279     private void grow(int minCapacity) {
280         // 保存之前老数组的长度
281         int oldCapacity=elementData.length;
282         // 扩展之前容量的一半
283         int newCapacity=oldCapacity+(oldCapacity>>1);
284         // 假如扩展之后还是比minCapacity小，就用minCapacity
285         if(newCapacity-minCapacity<0) {
286             newCapacity=minCapacity;
287         }
288         // 假如扩展之后比较MAX_ARRAY_SIZE大
289             // 需要通过minCapacity重新就是一个新的大小
290         if(newCapacity-MAX_ARRAY_SIZE>0) {
291             newCapacity=hugeCapacity(minCapacity);
292         }
293         // 创建新的数组：copy之前值，newCapacity
294         elementData=Arrays.copyOf(elementData, newCapacity);
295     }
296 
297     private int hugeCapacity(int minCapacity) {
298         // 假如minCapacity变成负数就抛OutOfMemoryError
299         if(minCapacity<0) {
300             throw new OutOfMemoryError();
301         }
302         return (minCapacity>MAX_ARRAY_SIZE ? Integer.MAX_VALUE:MAX_ARRAY_SIZE);
303     }
304 
305     private int calculateCapactiy(Object[] elementData, int minCapacity) {
306         // 判断当前数组是空数组，假如空数组的：取默认初始化容量和minCapacity最大值
307         // 假如不为空直接返回minCapacity
308         if(elementData==DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
```

```
            return Math.max(DEFAULT_CAPACITY, minCapacity);
        }
        return minCapacity;
    }

    /**
     * @return
     * 获取列表大小
     */
    public int size() {
        return size;
    }

    /**
     * @return
     *  判断列表是否为空
     */
    public boolean isEmpty() {
        return size==0;
    }

}

public static void main(String[] args) {
        ArrayList<String> list=new ArrayList<String>();
        list.add("刘备");
        list.add("关羽");
        list.add("张飞");
        for (int i = 0; i < list.size(); i++) {
            System.out.printf("%s ",list.get(i));
        }
        System.out.println("");
        System.out.println(list.size());
        System.out.println(list.isEmpty());
        list.remove(0);
        System.out.println(list.size());
        for (int i = 0; i < list.size(); i++) {
            System.out.printf("%s ",list.get(i));
        }
        System.out.println("");
```

```
        list.clear();
        System.out.println(list.size());
        System.out.println(list.isEmpty());

        list.add("刘备");
        list.add("关羽");
        list.add("张飞");
        for (int i = 0; i < list.size(); i++) {
            System.out.printf("%s ",list.get(i));
        }
    }
结果:
刘备 关羽 张飞
3
false
2
关羽 张飞
0
true
刘备 关羽 张飞

* 查看ArrayList

```

* 能够说出ArrayList和LinkedList区别

  * ArrayList：底层是用数组实现，查询快，增删慢

  * LinkedList：底层是用双向链表的，查询慢，增删快

```
* 查询:
 * ArrayList
    * elementData[index]
 * LinkedList
    * node(index).e
    * private Node<E> node(int index) {
        // 查找方式:
        if(index<(size>>1)) {// size>>1--等价size=size/2,使用位移性能会高
```

```
            // 假如这个下标在一半左边，就向前找
            // 从第一个开始向前查找：迭代查找
            Node<E> node=first;
            for (int i = 0; i < index; i++) {
                node=node.next;
            }
            return node;
        }else {
            // 假如这个下标在一半右边，就向后找
            Node<E> node=last;
            for (int i=size-1; i > index; i--) {
                node=node.prev;
            }
            return node;
        }
    }
* 添加
* ArrayList
    // 确保内部数组容量，可以添加元素
    * ensureCapacityInternal(size+1);
       * 可能会扩容，产生新的组
        * int newCapacity=oldCapacity+(oldCapacity>>1);
        * elementData=Arrays.copyOf(elementData, newCapacity);
    //给数组的下一个坐标赋值
    * elementData[size++]=e;
 * LinkedList
    * private void linkLast(E e) {
        // 1 先保存最后一个节点
        Node<E> l=last;
        // 2 构建需要添加的节点（它的前一个节点就是上一次最后节点）
        Node<E> newNode=new Node<E>(last,e,null);
        // 3 判断上一次最后节点是否空
        if(l==null) {
          // 假如为空，代表第一次添加,把新节点复制给first
          first=newNode;
        }else {
            // 假如不为空，上一次最后节点next指向新的节点
            last.next=newNode;
        }
        // 新的节点变成最后节点
```

```
        last = newNode;
        // size 加一
        size++;
    }
 * 删除
  * ArrayList
    * 数组复制
     * int numMoved=size-index-1;
       if(numMoved>0) {
           System.arraycopy(elementData, index+1, elementData, index, numMoved
       }
       // 释放最后一个资源
       elementData[size--]=null;// 让垃圾回收资源
   * LinkedList
    * private E unLink(int index) {
       // 1 获得下标对应节点
       Node<E> node=node(index);
       // 2 获得下标对应对象
       E e=node.e;
       // 3 获得下标对应节点前节点
       Node<E> prev=node.prev;
       // 4 获得下标对应节点后节点
       Node<E> next=node.next;
       // 5 判断prev是否null
       if(prev==null) {
           //假如prev为null意味着下标对应节点是第一个节点,
           //把下一个节点变成第一节点
           first=next;
       }else {
           // 重新指向
           prev.next=next;
           node.next=null;
       }
       // 6 判断next是否为null
       if(next==null) {
           //假如next为null意味着下标对应节点是最后一个节点
           //把上一个节点作为最后一个节点
           last=prev;
       }else {
           // 重新指向
```

```
            next.prev=prev;
            node.prev=null;
        }
        // 释放e
        node.e=null;
        // size 减一
        size--;
        return e;
    }
```