

## | 今天学习目标

- \* 能够理解数据结构之链表
- \* 能够实现自定义LinkedList
- \* 能够实现自定义ArrayList
- \* 能够说出ArrayList和LinkedList区别

---

## \* 回顾

### \* 数据结构：

- \* 存储，组织数据的方式
- \* 代码写更优雅，更高效

### \* 栈：FILO,Stack,方法栈

### \* 队列：FIFO,Queue,LinkedList

### \* 数组：长度不变，连续空间，增删慢

#### \* 稀疏数组

### \* 链表，LinkedList

## \* 能够理解数据结构之链表

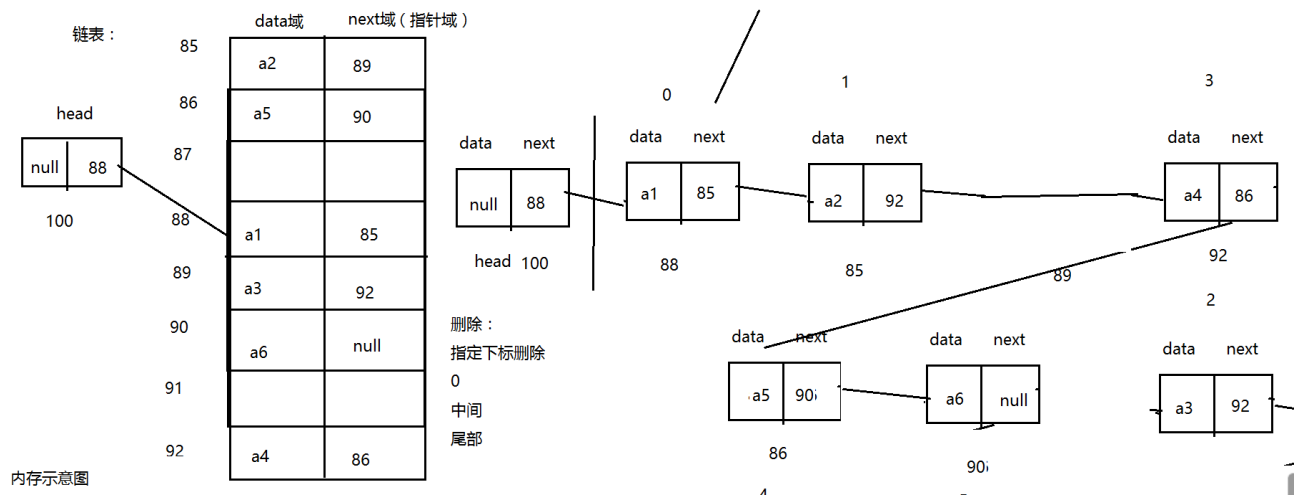
### \* 链表英文为：Linked List

\* 链表是由一系列结点node（链表中每一个元素称为结点）组成，结点可以在运行时动态生成。每个结点包括两个部分：一个是存储数据元素的数据域，另一个是存储下一个结点地址的指针域。我们常说链表有两种：单向链表和双向链表。

### \* 手拉手

- \* 链表可以带头节点也不带，看具体的需求
- \* 链表各个节点不一定是连续存储的。
- \* 多个结点之间，通过地址进行连接。
- \* 特点：查询慢，增删快。
- \* 单链表示意图

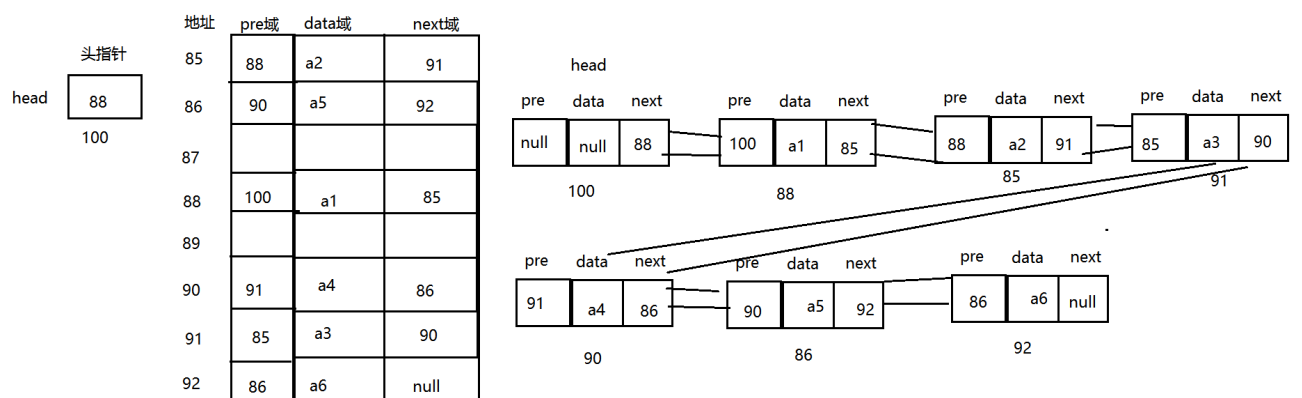




## \* 双向链表

\* 单向链表只能往一个方向查找。

\* 双向链表可以双向查找



## \* 能够实现自定义LinkedList

- 1 \* 需要实现功能
- 2 \* 添加节点（默认添加和指定位置添加）
- 3 `boolean add(E e)`
- 4 `boolean add(int index, E e)`
- 5 \* 访问某一个节点
- 6 `E get(int index);`
- 7 \* 删除节点
- 8 `E remove(int index);`
- 9 \* 获得链表的长度大小
- 10 `int size();`
- 11 \* 判断链表是否为空
- 12 `boolean isEmpty();`

```
13 * 清空链表
14 void clear();
15
16 思路:
17 * 新建LinkedList类
18 * 无参数构造器
19 * 新建LinkedList内部类Node
20 * 前一个节点（指针域）
21 * 当前节点的data
22 * 后一个节点（指针域）
23 * 构造器（三个参数的）
24 * 在LinkedList类
25 * 链表的大小
26 * 第一个节点
27 * 最后一个节点
28 * 获得链表的长度大小size()
29 * 直接返回size
30 * 判断链表是否为空
31 * 返回size==0;
32 * 添加节点（默认添加）boolean add(E e)
33 * 添加到最后方法编写linkLast(e);
34 * 先保存最后一个节点
35 * 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）
36 * 判断上一次最后节点是否空
37 * 假如为空，代表第一次添加,把新节点复制给first
38 * 假如不为空，上一次最后节点next指向新的节点
39 * 新的节点变成最后节点
40 * size 加一
41 * 直接返回true
42
43 * 添加节点（指定位置）
44 * 检测下标是否在这个链表添加合理位置
45 * checkPositionIndex
46 * isPositionIndex: 是否合理: 不合理抛IndexOutOfBoundsException
47 * index>=0 && index<=size
48 * 判断是否要加到最后还是插入到某个位置
49 * 判断条件: size==index
50 * 假如要加到最后: 调用 linkLast(e);
51 * 假如插入到某个位置
52 * 找到下标已经存在节点 (Node<E> node=node(index);)
```

```

53      * 判断条件: index < ( size>>1 )
54      * 假如这个下标在一半左边, 就向前找
55          * 从第一个开始向前查找: 迭代查找
56          Node<E> node=first;
57              for (int i = 0; i < index; i++) {
58                  node=node.next;
59              }
60          return node;
61      * 假如这个下标在一半右边, 就向后找
62          * 从最后一个向后查找: 迭代查找
63          Node<E> node=last;
64          for (int i=size-1; i > index; i--) {
65              node=node.prev;
66          }
67          return node;
68      * 链接到这个节点前面linkBefore
69          * 保存下标已经存在节点的前一个节点 (pred)
70          * 构建新的节点
71          * 新的节点重新指向
72          * 判断pred是否为null
73          * 假如pred为null意味下标已经存在节点的前一个节点为空, 意味要插入就是第一
74              * first=newNode;
75          * 假如pred不为null新的节点重新指向
76              * pred.next=newNode;
77          * size加一
78      * 返回true
79
80      * 访问某一个节点 (E get(int index))
81          * 检查下标是否在合理的范围 checkElementIndex
82              * isElementIndex : 假如不合理: IndexOutOfBoundsException
83              * 判断条件: index>=0 && index<size;
84          * 接通过下标查找节点: node
85      * 删除节点
86          * 检查下标是否在合理的范围: isElementIndex
87          * 断开连接并获取删除对象 E unlink(int index)
88          * 获得下标对应节点
89              * Node<E> node=node(index);
90          * 获得下标对应对象
91              * E e=node.e;
92          * 获得下标对应节点前节点

```

```

93         * Node<E> prev=node.prev;
94     * 获得下标对应节点后节点
95         * Node<E> next=node.next;
96     * 判断prev是否null
97         * 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
98         * first=next;
99         * 假如prev不为null: 重新指向
100         * prev.next=next;
101         node.next=null;
102     * 判断next是否为null
103         * 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
104         * last=prev;
105         * 假如next不为null: 重新指向
106         * next.prev=prev;
107         node.prev=null;
108     * 释放e
109         * node.e=null;
110     * size 减一
111         * size--;
112     * 返回e
113 * 清空链表 void clear
114     * 从第一个节点开始迭代
115         * for(Node<E> e=first;e!=null;)
116     * 先获取它下一节点保存起来，再释放资源
117         * Node<E> next=e.next;
118     * 释放资源
119         * e.e=null;
120         e.prev=null;
121         e.next=null;
122     * e 重新指向下一个
123         * e=next;
124     * first,last 释放资源
125         first=null;
126         last=null;
127     * size 重置为0
128         size=0;
129
130 代码:
131 public class LinkedList<E> {
132     private int size=0;// 链表的大小

```

```
133     private Node<E> first;// 第一个节点
134     private Node<E> last;// 最后一个节点
135
136     public LinkedList() {
137
138     }
139     /**
140      * @param e 添加的元素
141      * @return 是否添加成功
142      */
143     public boolean add(E e) {
144         // 添加到最后
145         linkLast(e);
146         return true;
147     }
148
149     public boolean add(int index,E e) {
150         // 1 检测下标是否在这个链表添加合理位置
151         checkPositionIndex(index);
152         // 2 判断是否要加到最后还是插入到某个位置
153         if(index==size) {
154             //加到最后
155             linkLast(e);
156         }else {
157             //插入到某个位置
158             // 找到下标已经存在节点
159             Node<E> node=node(index);
160             // 链接到这个节点前面
161             linkBefore(e,node);
162         }
163         return true;
164     }
165
166     private void linkBefore(E e, Node<E> node) {
167         //保存下标已经存在节点的前一个节点
168         Node<E> pred=node.prev;
169         //构建新的节点
170         Node<E> newNode=new Node<E>(pred,e,node);
171         // 新的节点重新指向
172         node.prev=newNode;
```

```

173         if(pred==null) {
174             // 下标已经存在节点的前一个节点为空,意味插入就是第一个简单
175             first=newNode;
176         }else {
177             pred.next=newNode;
178         }
179         // size++
180         size++;
181
182     }
183     private Node<E> node(int index) {
184         // 查找方式:
185         if(index<(size>>1)) { // size>>1--等价size=size/2,使用位移性能会高
186             // 假如这个下标在一半左边,就向前找
187             // 从第一个开始向前查找: 迭代查找
188             Node<E> node=first;
189             for (int i = 0; i < index; i++) {
190                 node=node.next;
191             }
192             return node;
193         }else {
194             // 假如这个下标在一半右边,就向后找
195             Node<E> node=last;
196             for (int i=size-1; i > index; i--) {
197                 node=node.prev;
198             }
199             return node;
200         }
201     }
202     private void checkPositionIndex(int index) {
203         if(!isPositionIndex(index)) {
204             throw new IndexOutOfBoundsException("Index:"+index+",Size:"+size);
205         }
206     }
207     private boolean isPositionIndex(int index) {
208         return index>=0 && index<=size;
209     }
210
211     /**
212     * @param index:需要查找下标的节点

```



```
213     * @return
214     */
215     public E get(int index) {
216         // 检查下标是否在合理的范围
217         checkElementIndex(index);
218         // 直接通过下标查找节点
219         return node(index).e;
220     }
221
222     private void checkElementIndex(int index) {
223         if(!isElementIndex(index)) {
224             throw new IndexOutOfBoundsException("Index:"+index+",Size:"+size);
225         }
226     }
227     private boolean isElementIndex(int index) {
228         return index>=0 && index<size;
229     }
230
231     public E remove(int index) {
232         // 检查下标是否在合理的范围
233         checkElementIndex(index);
234         // 断开连接并获取删除对象
235         return unLink(index);
236     }
237
238     /**
239     * @param index
240     * @return
241     * 需要删除对象，断开连接
242     */
243     private E unLink(int index) {
244         // 1 获得下标对应节点
245         Node<E> node=node(index);
246         // 2 获得下标对应对象
247         E e=node.e;
248         // 3 获得下标对应节点前节点
249         Node<E> prev=node.prev;
250         // 4 获得下标对应节点后节点
251         Node<E> next=node.next;
252         // 5 判断prev是否null
```

```

253     if(prev==null) {
254         //假如prev为null意味着下标对应节点是第一个节点,
255         //把下一个节点变成第一节点
256         first=next;
257     }else {
258         // 重新指向
259         prev.next=next;
260         prev.next=null;
261     }
262     // 6 判断next是否为null
263     if(next==null) {
264         //假如next为null意味着下标对应节点是最后一个节点
265         //把上一个节点作为最后一个节点
266         last=prev;
267     }else {
268         // 重新指向
269         next.prev=prev;
270         next.prev=null;
271     }
272     // 释放e
273     node.e=null;
274     // size 减一
275     size--;
276     return e;
277 }
278 /**
279  * @param e
280  */
281 private void linkLast(E e) {
282     // 1 先保存最后一个节点
283     Node<E> l=last;
284     // 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）
285     Node<E> newNode=new Node<E>(last,e,null);
286     // 3 判断上一次最后节点是否空
287     if(l==null) {
288         // 假如为空, 代表第一次添加,把新节点复制给first
289         first=newNode;
290     }else {
291         // 假如不为空, 上一次最后节点next指向新的节点
292         last.next=newNode;

```

```
293     }
294     // 新的节点变成最后节点
295     last = newNode;
296     // size 加一
297     size++;
298 }
299
300 /**
301  * 清空
302  */
303 public void clear() {
304     // 从第一个节点开始
305     for(Node<E> e=first;e!=null;) {
306         // 先获取它下一节点保存起来，再释放资源
307         Node<E> next=e.next;
308         // 释放资源
309         e.e=null;
310         e.prev=null;
311         e.next=null;
312         // e 重新指向下一个
313         e=next;
314     }
315     // first,last 释放资源
316     first=null;
317     last=null;
318     // size 重置为0
319     size=0;
320 }
321 /**
322  * @return
323  * 获得链表的长度大小
324  */
325 public int size() {
326     return size;
327 }
328
329 /**
330  * @return
331  * 判断链表是否为空
332  */
```

```

333     public boolean isEmpty() {
334         return size()==0;
335     }
336
337     /**
338      * @author xiaozhao
339      * 链接的节点
340      * @param <E>
341      */
342     private class Node<E>{
343         /**
344          * 前一个节点（指针域）
345          */
346         Node<E> prev;
347         /**
348          * 当前节点的data
349          */
350         E e;
351         /**
352          * 后一个节点（指针域）
353          */
354         Node<E> next;
355         public Node(Node<E> prev, E e, Node<E> next) {
356             super();
357             this.prev = prev;
358             this.e = e;
359             this.next = next;
360         }
361     }
362 }
363
364
365     public static void main(String[] args) {
366         LinkedList<String> list=new LinkedList<String>();
367         list.add("刘备");
368         list.add("关羽");
369         list.add("张飞");
370         list.add(1,"关羽11");
371         System.out.println(list.size());
372         System.out.println(list.get(0));

```

```

373     System.out.println(list.get(1));
374     System.out.println(list.get(2));
375     System.out.println(list.get(3));
376     System.out.println(list.isEmpty());
377     System.out.println(list.remove(0));
378     System.out.println(list.size());
379     list.clear();
380     System.out.println(list.isEmpty());
381     System.out.println(list.size());
382 }
383 * 结果
384 4
385 刘备
386 关羽11
387 关羽
388 张飞
389 false
390 刘备
391 3
392 true
393 0
394
395 * 查看LinkedList部分源码

```

自定义LinkedList  
添加功能  
size();  
isEmpty();  
clear();  
add(E e)  
add(int index,E e);  
E get(int index)  
remove(int index)

```

* boolean add(E e) : 默认是添加到链表尾部
* linkLast
* 弄变量保存临时last--f
* 构建新的节点，等着被添加
    Node newNode=new Node(last,e,null);
* 判断这个f是否为null
* 假如f为null;这个newNode就要作为第一个节点
    first=newNode
* 假如f不为null
    last.next=newNode;
* newNode会成为最后节点
    last=newNode
* size++ ;
* return true;

```

```

* add ( int index,E e )
* 检测这个下标是否能添加 checkIndex...
    * index>0 && index<=size
    * index<0 || index>size
* 添加有三种情况
* 添加到末尾
    if(index==size){linkLast(index); else{
        * 找到需要添加节点下标
        Node node=node ( index ) ;
        * linkBefore(e,node)
        * Node prev=node.prev;
        * Node next=node.next;
        * Node newNode=new Node(prev,e,next);
        * node.prev=newNode;
        if(prev==null){
            first=newNode;
        }else{ prev.next=newNode;
        }
    }
}

```

```

* E remove(int index )
* 检测下标
* 找到需要删除节点
    Node node=node ( index ) ;
    Node prev=node.prev;
    Node next=node.next;
    E e=node.e;
    if(prev==null){ first=next; }else{
        prev.next=next;node.prev=null; }
    if(next==null){last=prev;}else{
        next.prev=prev;node.next=null
    }
    node.e=null;
    size--;
    return e;

```

\* 能够实现自定义ArrayList

\* 能够说出ArrayList和LinkedList区别