简单集合：

ArrayList、HashMap、LinkedList

LinkedList 是一个继承于AbstractSequentialList的双向链表。它也可以被当作堆栈、队列或双端队列进行操作。同时实现了Deque接口，即能将LinkedList当作双端队列使用。

另外实现了Cloneable接口，Serializable接口。

接下来看LinkedList的构造方法：

```
/**
     * Constructs an empty list.
     */
    public LinkedList() {
    }

    /**
     * Constructs a list containing the elements of the specified
     * collection, in the order they are returned by the collecti
on's
     * iterator.
     *
     * @param c the collection whose elements are to be placed i
nto this list
     * @throws NullPointerException if the specified collection
is null
     */
    public LinkedList(Collection<? extends E> c) {
        this();
        addAll(c);
    }
```

LinkedList提供了两个构造方法，第一个是默认无参的，第二个是带Collection的类型参数：

- 使用this()调用默认的构造方法

成员变量分析

```
/**
     * 当前有多少个节点
```

```
     */
transient int size = 0;
    /**
     * 第一个节点.
     * Invariant: (first == null && last == null) ||
     *            (first.prev == null && first.item != null)
     */
    transient Node<E> first;

    /**
     * 最后一个节点.
     * Invariant: (first == null && last == null) ||
     *            (last.next == null && last.item != null)
     */
    transient Node<E> last;
```

核心方法分析

1. addAll()方法

addAll有两个重载函数，addAll(Collection<? extends E>)型和addAll(int, Collection<? extends E>)型，我们平时习惯调用的addAll(Collection<? extends E>)型会转化为addAll(int, Collection<? extends E>)型，所以我们着重分析此函数即可

```
public boolean addAll(int index, Collection<? extends E> c) {
//JDK8将对index的判断封装了一个方法checkPositionIndex(index);
//这个就不用说了，集合转为数组
        Object[] a = c.toArray();
        int numNew = a.length;
        if (numNew == 0)
            return false;
//succ指向当前需要插入节点的位置，pred指向其前一个节点
        Node<E> pred, succ;
 //在列表尾部插入的时候
         if (index == size) {
             succ = null;
             pred = last;
```

```
        } else {
//若不是在尾部插入时候则先去根据索引查询对应的元素可见该块最下面的node()方法
            succ = node(index);
            pred = succ.prev;
        }
//遍历collection中的所有元素将其依次插入到此链表中指定位置
        for (Object o : a) {
            @SuppressWarnings("unchecked") E e = (E) o;
            Node<E> newNode = new Node<>(pred, e, null);
            if (pred == null)
                first = newNode;
            else
                pred.next = newNode;
            pred = newNode;
        }

        if (succ == null) {
            last = pred;
        } else {
            pred.next = succ;
            succ.prev = pred;
        }

        size += numNew;
        modCount++;
        return true;
    }

/**
     * 根据index返回对应元素.
     */
    Node<E> node(int index) {
        // assert isElementIndex(index);
//若index<size/2正序移位获取索引位置
```

```
        if (index < (size >> 1)) {
            Node<E> x = first;
            for (int i = 0; i < index; i++)
                x = x.next;
            return x;
        } else {
            Node<E> x = last;
            for (int i = size - 1; i > index; i--)
                x = x.prev;
            return x;
        }
    }
```

2. removeXXX()方法

LinkedList提供了头删除removeFirst()、尾删除removeLast()、remove(int index)、remove(Object o)、clear()这些删除元素的方法。

```
/**
     * Removes and returns the first element from this list.
     */
public E removeFirst() {
        final Node<E> f = first;
        if (f == null)
            throw new NoSuchElementException();
        return unlinkFirst(f);
    }

    /**
     * Removes and returns the last element from this list.
     */
    public E removeLast() {
        final Node<E> l = last;
        if (l == null)
            throw new NoSuchElementException();
        return unlinkLast(l);
    }
```

```
20 /**
21      * Unlinks non-null first node f.
22      * 删除非空的首节点f.
23      */
24     private E unlinkFirst(Node<E> f) {
25         // assert f == first && f != null;
26         final E element = f.item;
27         final Node<E> next = f.next;
28         f.item = null;
29         f.next = null; // help GC
30 //将原首节点的next节点设置为首节点
31         first = next;
32         if (next == null)
33             last = null;
34         else
35             next.prev = null;
36         size--;
37         modCount++;
38         return element;
39     }
40 
41     /**
42      * Unlinks non-null last node l.
43      * 删除非空的尾节点f.
44      */
45     private E unlinkLast(Node<E> l) {
46         // assert l == last && l != null;
47         final E element = l.item;
48         final Node<E> prev = l.prev;
49         l.item = null;
50         l.prev = null; // help GC
51 //将原尾节点的prev节点设置为尾节点
52         last = prev;
53         if (prev == null)
54             first = null;
```

```
55          else
56              prev.next = null;
57          size--;
58          modCount++;
59          return element;
60      }
61
62
63  /**
64       * Remove.
65       */
66  public boolean remove(Object o) {
67          if (o == null) {
68              for (Node<E> x = first; x != null; x = x.next) {
69                  if (x.item == null) {
70                      unlink(x);
71                      return true;
72                  }
73              }
74          } else {
75              for (Node<E> x = first; x != null; x = x.next) {
76                  if (o.equals(x.item)) {
77                      unlink(x);
78                      return true;
79                  }
80              }
81          }
82          return false;
83      }
84  /**
85       * Unlinks non-null node x.
86       * 删除非空节点.
87       */
88      E unlink(Node<E> x) {
89          // assert x != null;
```

```
90          final E element = x.item;
91          final Node<E> next = x.next;
92          final Node<E> prev = x.prev;
93  //如果被删除节点为头节点
94          if (prev == null) {
95              first = next;
96          } else {
97              prev.next = next;
98              x.prev = null;
99          }
100 //如果被删除节点为尾节点
101          if (next == null) {
102              last = prev;
103          } else {
104              next.prev = prev;
105              x.next = null;
106          }
107 
108          x.item = null;
109          size--; //size-1
110          modCount++;
111          return element;
112     }
113 
114 /**
115      * Removes all of the elements from this list.
116      * 清空所有节点.
117      */
118     public void clear() {
119         // Clearing all of the links between nodes is "unnecessa
ry", but:
120         // - helps a generational GC if the discarded nodes inha
bit
121         //   more than one generation
122         // - is sure to free memory even if there is a reachable
Iterator
```

```
        for (Node<E> x = first; x != null; ) {
            Node<E> next = x.next;
            x.item = null;
            x.next = null;
            x.prev = null;
            x = next;
        }
        first = last = null;
        size = 0;
        modCount++;
    }
```

3. set()方法

//很容易分析，先检查index，然后根据index返回对应元素，最后将元素-->x.item

```
public E set(int index, E element) {
        checkElementIndex(index);
        Node<E> x = node(index);
        E oldVal = x.item;
        x.item = element;
        return oldVal;
    }
```

4. getXXX()方法

LinkedList提供了getFirst()、getLast()、contains(Object o)、get(int index)、indexOf(Object o)、lastIndexOf(Object o)这些查找元素的方法。

```
/**
     * Returns the first element in this list.
     */
    public E getFirst() {
        final Node<E> f = first;
        if (f == null)
            throw new NoSuchElementException();
        return f.item;
    }
/**
     * Returns the last element in this list.
```

```
     */
    public E getLast() {
        final Node<E> l = last;
        if (l == null)
            throw new NoSuchElementException();
        return l.item;
    }
public boolean contains(Object o) {
        return indexOf(o) != -1;
    }
/**
     * 正向查找，返回LinkedList中元素值Object o第一次出现的位置，如果元素不存在，则返回-1
     */
    public int indexOf(Object o) {
        int index = 0;
        if (o == null) {
            for (Node<E> x = first; x != null; x = x.next) { //正向
                if (x.item == null)
                    return index;
                index++;
            }
        } else {
            for (Node<E> x = first; x != null; x = x.next) {
                if (o.equals(x.item))
                    return index;
                index++;
            }
        }
        return -1;
    }
//逆向查找，返回LinkedList中元素值Object o最后一次出现的位置，如果元素不存在，则返回-1
    public int lastIndexOf(Object o) {
        int index = size;
```

```
//LinkedList可以为null
        if (o == null) {
            for (Node<E> x = last; x != null; x = x.prev) { //逆向
                index--;
                if (x.item == null)
                    return index;
            }
        } else {
            for (Node<E> x = last; x != null; x = x.prev) {
                index--;
                if (o.equals(x.item))
                    return index;
            }
        }
        return -1;
    }
/**
     * 根据index获取当前元素.
     */
    public E get(int index) {
        checkElementIndex(index);
        return node(index).item;
    }
```

5. Queue操作

Queue操作提供了peek()、element()、poll()、remove()、offer(E e)这些方法。

```

//获取但不移除此队列的头；如果此队列为空，则返回 null
    public E peek() {
        final Node<E> f = first;
        return (f == null) ? null : f.item;
    }

    //获取但不移除此队列的头；如果此队列为空，则抛出NoSuchElementException异常
```

```
    public E element() {
        return getFirst();
    }

    //获取并移除此队列的头，如果此队列为空，则返回 null
    public E poll() {
        final Node<E> f = first;
        return (f == null) ? null : unlinkFirst(f);
    }

    //获取并移除此队列的头，如果此队列为空，则抛出NoSuchElementException异常
    public E remove() {
        return removeFirst();
    }

    //将指定的元素值(E e)插入此列表末尾
    public boolean offer(E e) {
        return add(e);
    }
```

6. Deque操作

Deque操作提供了offerFirst(E e)、offerLast(E e)、peekFirst()、peekLast()、pollFirst()、pollLast()、push(E e)、pop()、removeFirstOccurrence(Object o)、removeLastOccurrence(Object o)这些方法。

```
//将指定的元素值(E e)插入此列表末尾
    public boolean offer(E e) {
        return add(e);
    }

    // Deque operations

    //将指定的元素插入此双端队列的开头
    public boolean offerFirst(E e) {
        addFirst(e);
```

```
        return true;
    }

    //将指定的元素插入此双端队列的末尾
    public boolean offerLast(E e) {
        addLast(e);
        return true;
    }

    //获取，但不移除此双端队列的第一个元素；如果此双端队列为空，则返回 null
    public E peekFirst() {
        final Node<E> f = first;
        return (f == null) ? null : f.item;
     }

    //获取，但不移除此双端队列的最后一个元素；如果此双端队列为空，则返回 null
    public E peekLast() {
        final Node<E> l = last;
        return (l == null) ? null : l.item;
    }

    //获取并移除此双端队列的第一个元素；如果此双端队列为空，则返回 null
    public E pollFirst() {
        final Node<E> f = first;
        return (f == null) ? null : unlinkFirst(f);
    }

    //获取并移除此双端队列的最后一个元素；如果此双端队列为空，则返回 null
    public E pollLast() {
        final Node<E> l = last;
        return (l == null) ? null : unlinkLast(l);
    }
```

```
43
44      //将一个元素推入此双端队列所表示的堆栈（换句话说，此双端队列的头部）
45      public void push(E e) {
46          addFirst(e);
47      }
48
49      //从此双端队列所表示的堆栈中弹出一个元素（换句话说，移除并返回此双端队列的头部）
50      public E pop() {
51          return removeFirst();
52      }
53
54      //从此双端队列移除第一次出现的指定元素，如果列表中不包含次元素，则没有任何改变
55      public boolean removeFirstOccurrence(Object o) {
56          return remove(o);
57      }
58
59      //从此双端队列移除最后一次出现的指定元素,如果列表中不包含次元素，则没有任何改变
60      public boolean removeLastOccurrence(Object o) {
61          //由于LinkedList中允许存放null，因此下面通过两种情况来分别处理
62          if (o == null) {
63              for (Node<E> x = last; x != null; x = x.prev) { //逆向向前
64                  if (x.item == null) {
65                      unlink(x);
66                      return true;
67                  }
68              }
69          } else {
70              for (Node<E> x = last; x != null; x = x.prev) {
71                  if (o.equals(x.item)) {
72                      unlink(x);
73                      return true;
```

```
74                 }
75             }
76         }
77          return false;
78     }
79
```