LinkedList底层是基于链表实现，所以链表的一些特征在ListedList中同样适用，链表中为了能够进行节点连接，在节点（Node）中增加属性**prev**和**next**进行上下游串联。本文按照先分析节点（Node）结构，后分析LInkedList中常用操作。



## 一、LinkedList中节点（Node）结构介绍

源码如下：

```
private static class Node<E> {
    E item;// 当前节点信息
    Node<E> next;// 下个节点引用
    Node<E> prev;// 上个节点引用

    Node(Node<E> prev, E element, Node<E> next) {
        this.item = element;
        this.next = next;
        this.prev = prev;
    }
}
```

新增内部类Node, 属性如下：

| 属性名 | 含义 | 备注 |
| --- | --- | --- |
| E **item** | 存储当前节点数据 | |
| Node<E> **next** | 记录下个节点的引用，用来进行数据连接 | |
| Node<E> **prev** | 记录上个节点的引用，用来进行数据连接 | |

## 二、插入(add)操作过程

### 1.尾部添加

```
/**
 * Appends the specified element to the end of this list.
 *
 * <p>This method is equivalent to {@link #addLast}.
 *
 * @param e element to be appended to this list
 * @return {@code true} (as specified by {@link Collection#add})
 */
public boolean add(E e) {
    linkLast(e);
    return true;
}
```

调用linkLast(e)

```
/**
 * Links e as last element.
 */
void linkLast(E e) {
    final Node<E> l = last;// 获取尾节点
    final Node<E> newNode = new Node<>(l, e, null);// 把当前节点追加到尾节点上，prev
    last = newNode// 把新节点复制给尾节点;
    if (l == null)// 如果尾节点为空，那么表示当前并没有元素，那么头节点也是当前节点
        first = newNode;
     else
        l.next = newNode;
     size++;
     modCount++;
}
```

**2.指定索引位置进行添加**

```
/**
 * Inserts the specified element at the specified position in this list.
 * Shifts the element currently at that position (if any) and any
 * subsequent elements to the right (adds one to their indices).
 *
 * @param index index at which the specified element is to be inserted
 * @param element element to be inserted
 * @throws IndexOutOfBoundsException {@inheritDoc}
 */
public void add(int index, E element) {
   // 检查索引位置是否合法 (index >= 0 & index <= size)
    checkPositionIndex(index);

  // 如果index == size,表示实在尾部添加
    if (index == size)
        linkLast(element);
    else
        linkBefore(element, node(index));// 在前面数据项进行添加
}
```

添加时进行校验index值的合法性，源代码如下：

```
/**
 * Tells if the argument is the index of a valid position for an
 * iterator or an add operation.
 */
private boolean isPositionIndex(int index) {
    return index >= 0 && index <= size;
}
```

指定位置进行添加时，先判断index值的合法性（index >= 0 & index <= size），如果index == size,表示实在尾部添加，否则是在前面数据项进行添加。

先找到index所对应的元素，调用方法node(index)，源代码如下

```
/**
```

```
 * Returns the (non-null) Node at the specified element index.
 */
Node<E> node(int index) {
    // assert isElementIndex(index);

    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;
    }
}
```

当前因为linkedList没有记录index信息，所以只能遍历查找（o(n/2)），当前查找方法为二分查找法，如果当前index值小于size的一半（index < (**size** >> 1)），那么从头部开始遍历，否则从尾部节点开始遍历。

linkBefore(E e, Node<E> succ) 方法源码：

```
/**
 * Inserts element e before non-null Node succ.
 */
void linkBefore(E e, Node<E> succ) {
    // assert succ != null;
    final Node<E> pred = succ.prev;
    final Node<E> newNode = new Node<>(pred, e, succ);
    succ.prev = newNode;
    if (pred == null)
        first = newNode;
    else
        pred.next = newNode;
    size++;
    modCount++;
}
```

举例子说明：

index对应节点(B), 当前节点(e)进行插入，比如原来顺序 A <-> B <-> C,那么插入后的顺序为 A <-> e <-> B <-> C.

## 三、删除操作remove(**int** index)

```
/**
 * Removes the element at the specified position in this list.  Shifts any
 * subsequent elements to the left (subtracts one from their indices).
 * Returns the element that was removed from the list.
 *
 * @param index the index of the element to be removed
 * @return the element previously at the specified position
 * @throws IndexOutOfBoundsException {@inheritDoc}
 */
public E remove(int index) {
    checkElementIndex(index);// 先坚持index值是否合法
    return unlink(node(index));// 进行删除操作
```

```
}
```

具体的删除操作在unlink方法中，代码如下：

```
/**
 * Unlinks non-null node x.
 */
E unlink(Node<E> x) {
    // assert x != null;
    final E element = x.item;
    final Node<E> next = x.next;
    final Node<E> prev = x.prev;

    if (prev == null) {
        first = next;
    } else {
        prev.next = next;
        x.prev = null;
    }

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

    x.item = null;
    size--;
    modCount++;
    return element;
}
```

举例子说明：

index对应节点(B)进行删除，比如原来顺序 A <-> B <-> C,那么插入后的顺序为 A <-> C

## 四、查找get(int index)

```
/**
 * Returns the element at the specified position in this list.
 *
 * @param index index of the element to return
 * @return the element at the specified position in this list
 * @throws IndexOutOfBoundsException {@inheritDoc}
 */
public E get(int index) {
   // 坚持index值是否合法
     checkElementIndex(index);
    // 二分查找
     return node(index).item;
}
```

根据index 与当前size值的大小，进行二分查找。