**Java LinkedList的ListIterator**

@author ixenos

**ListIterator<E>是继承自Iterator<E>的接口**

　　故，ListIterator注意点：

　　1.迭代器**不存储所有元素的引用，只有两个指针，一个指向上一个返回得到的元素，另一个下一个未涉足的元素；**

　　2.迭代开始前先同步内外modCount，迭代过程中检查是否同步，如果外部结构改变，则迭代**快速失败(fast-fails机制)；**

3.ListIterator可以算半个LinkedList视图了，因为在迭代的途中还可以修改外部结构（add、remove）

**listIterator(int index)源码分析**

**listIterator**

public [ListIterator](http://www.cnblogs.com/)<[E](http://www.cnblogs.com/)> **listIterator**(int index)

返回此列表中的元素的列表迭代器（按适当顺序），**从列表中指定位置开始**。

**参数：**

index - 要从列表迭代器返回的第一个元素的索引（通过调用 next 方法）

**返回：**

此列表中的元素的 ListIterator（按适当顺序），从列表中指定位置开始

**抛出：**

[IndexOutOfBoundsException](http://www.cnblogs.com/) - 如果索引超出范围 (index < 0 || index > size())

**通过：**

内部类private class**ListItr** implements ListIterator<E> 实现

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 //迭代器不存储数值！是每迭代一次返回一次

2 // LinkedList中功能强大的ListIterator方法

3

4 public ListIterator<E> listIterator(int index) {

5 checkPositionIndex(index);

6 return new ListItr(index); //调用内部类ListItr的匿名对象

7 }

8

9 //把ListIterator接口送给内部类实现是为了与Iterator接口兼容，因为ListIterator接口继承自Iterator接口

10 private class ListItr implements ListIterator<E> { //实现了ListIterator接口

11 private Node<E> lastReturned; //指向上一个返回得到的元素

12 private Node<E> next; //指向下一个未涉足的元素

13 private int nextIndex;

14 private int expectedModCount = modCount;

15

16 ListItr(int index) {

17 // assert isPositionIndex(index);

18 next = (index == size) ? null : node(index);

19 nextIndex = index;

20 }

21

22 public boolean hasNext() {

23 return nextIndex < size;

24 }

25

26 public E next() {

27 checkForComodification();

28 if (!hasNext())

29 throw new NoSuchElementException();

30

31 lastReturned = next;

32 next = next.next;

33 nextIndex++;

34 return lastReturned.item;

35 }

36

37 public boolean hasPrevious() {

38 return nextIndex > 0;

39 }

40

41 public E previous() {

42 checkForComodification();

43 if (!hasPrevious())

44 throw new NoSuchElementException();

45

46 lastReturned = next = (next == null) ? last : next.prev;

47 nextIndex--;

48 return lastReturned.item;

49 }

50

51 public int nextIndex() {

52 return nextIndex;

53 }

54

55 public int previousIndex() {

56 return nextIndex - 1;

57 }

58

59 //可以删除哟

60 public void remove() {

61 checkForComodification(); //先确定外部modCount没变

62 if (lastReturned == null)

63 throw new IllegalStateException();

64

65 Node<E> lastNext = lastReturned.next;

66 unlink(lastReturned);

67 if (next == lastReturned)

68 next = lastNext;

69 else

70 nextIndex--;

71 lastReturned = null;

72 expectedModCount++; //删除外部元素modCount++所以内部的expectedModCount也++来同步

73 }

74

75 public void set(E e) {

76 if (lastReturned == null) //先确定外部modCount没变

77 throw new IllegalStateException();

78 checkForComodification();

79 lastReturned.item = e;

80 }

81

82 public void add(E e) {

83 checkForComodification();

84 lastReturned = null;

85 if (next == null)

86 linkLast(e); //如果next指针在队尾则直接加在队尾

87 else

88 linkBefore(e, next); //否则插入到next指针指向元素的前面

89 nextIndex++;

90 expectedModCount++; //删除外部元素modCount++所以内部的expectedModCount也++来同步

91

92 }

93

94 public void forEachRemaining(Consumer<? super E> action) {

95 Objects.requireNonNull(action); //如果action为空则抛出空指针异常

96 while (modCount == expectedModCount && nextIndex < size) {

97 action.accept(next.item);

98 lastReturned = next;

99 next = next.next;

100 nextIndex++;

101 }

102 checkForComodification();

103 }

104

105 //外部结构修改则迭代快速失败fast-fails

106 final void checkForComodification() {

107 if (modCount != expectedModCount)

108 throw new ConcurrentModificationException();

109 }

110 }

[复制代码](javascript:void(0);)

**利用ListItr实现的降序迭代**

**descendingIterator**

public [Iterator](http://www.cnblogs.com/)<[E](http://www.cnblogs.com/)> **descendingIterator**()

返回以逆向顺序在此双端队列的元素上进行迭代的迭代器。元素将按从最后一个（尾部）到第一个（头部）的顺序返回。

**实质就是在关键操作修改了指针**

**返回：**

以逆向顺序在此双端队列中的元素上进行迭代的迭代器

**从以下版本开始：**

1.6

http://images.cnblogs.com/OutliningIndicators/ExpandedBlockStart.gif

[复制代码](javascript:void(0);)

1 /\*\*降序迭代

2 \* @since 1.6

3 \*/

4 // sort 是顺序ascending 表示升descending 表示降

5 public Iterator<E> descendingIterator() {

6 return new DescendingIterator();

7 }

8

9 /\*\*

10 \* Adapter to provide descending iterators via ListItr.previous

11 \*/

12 //利用LisItr实现降序迭代

13 private class DescendingIterator implements Iterator<E> {

14 private final ListItr itr = new ListItr(size());

15 public boolean hasNext() {

16 return itr.hasPrevious(); //利用原有方法，改造了一下return

17 }

18 public E next() {

19 return itr.previous(); //利用原有方法，改造了一下return

20 }

21 public void remove() {

22 itr.remove();

23 }

24 }