**//迭代器不存储数值！是每迭代一次返回一次**

**// LinkedList中功能强大的ListIterator方法**

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

checkPositionIndex(index);

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

}

**//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**

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

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

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

**private** **int** nextIndex;

**private** **int** expectedModCount = modCount;

ListItr(**int** index) {

// assert isPositionIndex(index);

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

nextIndex = index;

}

**public** **boolean** hasNext() {

**return** nextIndex < size;

}

**public** E next() {

checkForComodification();

**if** (!hasNext())

**throw** **new** NoSuchElementException();

lastReturned = next;

next = next.next;

nextIndex++;

**return** lastReturned.item;

}

**public** **boolean** hasPrevious() {

**return** nextIndex > 0;

}

**public** E previous() {

checkForComodification();

**if** (!hasPrevious())

**throw** **new** NoSuchElementException();

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

nextIndex--;

**return** lastReturned.item;

}

**public** **int** nextIndex() {

**return** nextIndex;

}

**public** **int** previousIndex() {

**return** nextIndex - 1;

}

//可以删除哟

**public** **void** remove() {

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

**if** (lastReturned == **null**)

**throw** **new** IllegalStateException();

Node<E> lastNext = lastReturned.next;

unlink(lastReturned);

**if** (next == lastReturned)

next = lastNext;

**else**

nextIndex--;

lastReturned = **null**;

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

}

**public** **void** set(E e) {

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

**throw** **new** IllegalStateException();

checkForComodification();

lastReturned.item = e;

}

**public** **void** add(E e) {

checkForComodification();

lastReturned = **null**;

**if** (next == **null**)

linkLast(e); **//如果next指针在队尾则直接加在队尾**

**else**

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

nextIndex++;

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

}

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

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

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

action.accept(next.item);

lastReturned = next;

next = next.next;

nextIndex++;

}

checkForComodification();

}

//外部结构修改则迭代快速失败fast-fails

**final** **void** checkForComodification() {

**if** (modCount != expectedModCount)

**throw** **new** ConcurrentModificationException();

}

}

/\*\*降序迭代

\* **@since** 1.6

\*/

// sort 是顺序ascending 表示升descending 表示降

**public** Iterator<E> descendingIterator() {

**return** **new** DescendingIterator();

}

/\*\*

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

\*/

//利用LisItr实现降序迭代

**private** **class** DescendingIterator **implements** Iterator<E> {

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

**public** **boolean** hasNext() {

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

}

**public** E next() {

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

}

**public** **void** remove() {

itr.remove();

}

}