　　　　序言

　　　　　　　　写的ArrayList源码分析这篇文章，第一次登上首页，真是有点开心啊，再接再厉。这只是第一步，希望以后写的文章更多的登上首页，让更多的人看到，共同学习，能帮助到别人就最好不过了。开始这一系列的第二篇文章吧，LinkedList。

　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　　--WH

补充内容

　　　　　　首先要知道一种数据结构，就是链表，什么样的形式是链表呢？推荐博文：[http://www.jianshu.com/p/681802a00cdf](http://www.jianshu.com/p/681802a00cdf" \o "http://www.jianshu.com/p/681802a00cdf" \t "http://www.cnblogs.com/whgk/p/_blank)

　　　　　　　　　　注意：上面推荐博文中的大部分是可以参考的，但是他写这篇博文的时候，是在很久之前了，所以他里面介绍的linkedList是一个双向循环链表，但是现在我写的版本是JDK1.7了，通过

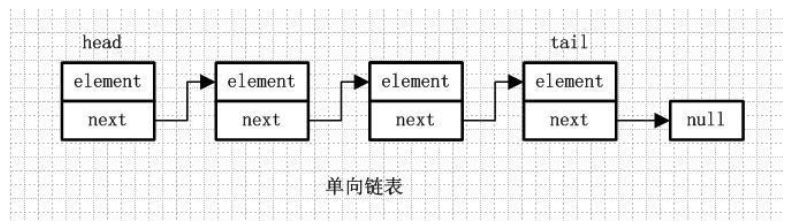
查看源码，发现linkedList已经不是双向循环链表，只是一个双向链表，这里请分清楚，别搞懵了。如果还不知道我在说什么，就请先看完我这篇文章然后再回过头来看这句话。

　　　　　　1、单向链表:

　　　　　　　　　　element：用来存放元素

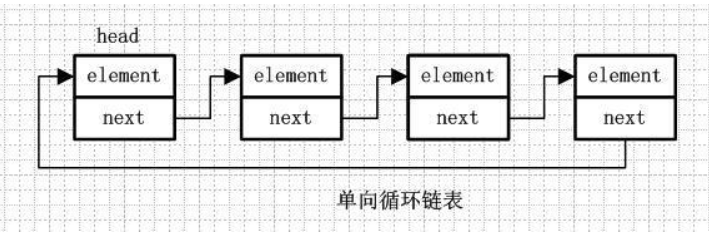
　　　　　　　　　　next：用来指向下一个节点元素

　　　　　　　　　　通过每个结点的指针指向下一个结点从而链接起来的结构，最后一个节点的next指向null。

\

　　　　　　2、单向循环链表：element、next 跟前面一样。

　　　　　　　　　　在单向链表的最后一个节点的next会指向头节点，而不是指向null，这样存成一个环



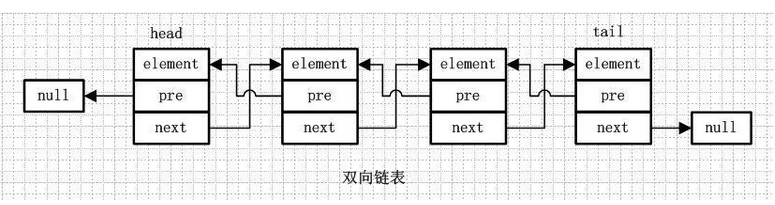
　　　　　　3、双向链表：

　　　　　　　　　　element：存放元素

　　　　　　　　　　pre：用来指向前一个元素

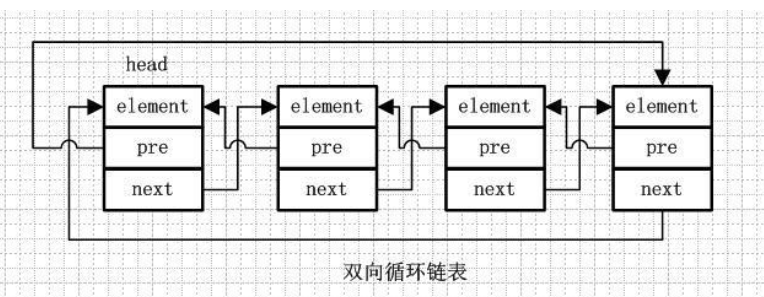
　　　　　　　　　　next：指向后一个元素

　　　　　　　　　　双向链表是包含两个指针的，pre指向前一个节点，next指向后一个节点，但是第一个节点head的pre指向null，最后一个节点的tail指向null。



　　　　　　4、双向循环链表：element、pre、next

　　　　　　　　　　第一个节点的pre指向最后一个节点，最后一个节点的next指向第一个节点，也形成一个“环”。



一、初识LinkedList

　　　　　　在我们平常用linkedList时，我们肯定都知道linkedList的一些特点

　　　　　　　　　　1、是通过链表实现的，

　　　　　　　　　　2、如果在频繁的插入，或者删除数据时，就用linkedList性能会更好。

　　　　　　我们只知道这些特性只是很片面的，并且只是常识，现在通过我们自己去查api，看官方是如何解释LinkedList的。(版本jdk1.7)

　　　　　　　　　　1、Doubly-linked list implementation of the List and Deque interfaces. Implements all optional list operations, and permits all elements (including null).

　　　　　　　　　　　　　　这告诉我们，linkedList是一个双向链表，并且实现了List和Deque接口中所有的列表操作，并且能存储任何元素，包括null，这里我们可以知道linkedList除了可以当链表使用，还可以当作队列使用，并能进行相应的操作。

　　　　　　　　　　2、All of the operations perform as could be expected for a doubly-linked list. Operations that index into the list will traverse the list from the beginning or the end, whichever is closer to the specified index.

　　　　　　　　　　　　　　这个告诉我们，linkedList在执行任何操作的时候，都必须先遍历此列表来靠近通过index查找我们所需要的的值。通俗点讲，这就告诉了我们这个是顺序存取，每次操作必须先按开始到结束的顺序遍历，随机存取，就是arrayList，能够通过index。随便访问其中的任意位置的数据，这就是随机列表的意思。

　　　　　　　　　　3、api中接下来讲的一大堆，就是说明linkedList是一个非线程安全的(异步)，其中在操作Interator时，如果改变列表结构(add\delete等)，会发生fail-fast。

　　　　　　　通过查看api，重新总结一下目前我们知道的linkedList的特性

　　　　　　　　　　　1、异步，也就是非线程安全

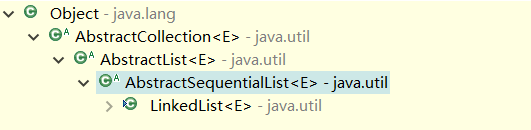
　　　　　　　　　　　2、双向链表、由于实现了list和Deque接口，能够当作队列来使用并且又是双向链表那么就有着链表的好处和坏处。

　　　　　　　　　　　　　　　　链表：查询效率不高，但是插入和删除这种操作性能好。

　　　　　　　　　　   3、是顺序存取结构，注意和随机存取结构两个概念搞清楚

二、LinkedList的继承结构以及层次关系

　　　　　　1、继承结构：



 　　　　　　　　我们可以看到，linkedList在最底层，说明他的功能最为强大，并且细心的还会发现，arrayList只有四层，这里多了一层AbstractSequentialList的抽象类，为什么呢？不知道就查api，api文档中都会有说明的

　　　　　　　　　　　　·减少实现顺序存取（例如LinkedList）这种类的工作，通俗的讲就是方便，抽象出类似LinkedList这种类的一些共同的方法

　　　　　　　　　　　　·既然有了上面这句话，那么以后如果自己想实现顺序存取这种特性的类(就是链表形式)，那么就继承这个AbstractSequentialList抽象类，如果想像数组那样的随机存取的类，那么就去实现AbstracList抽象类，

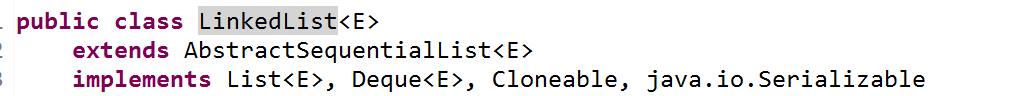
　　　　　　　　　　　　·这样的分层，就很符合我们抽象的概念，越在高处的类，就越抽象，往在底层的类，就越有自己独特的个性。自己要慢慢领会这种思想。

[IMG_261](http://www.cnblogs.com/whgk/p/javascript:void(0);)

1 public abstract class AbstractSequentialList<E> 2 extends AbstractList<E> 3 //这里第一段就解释了这个类的作用，这个类为实现list接口提供了一些重要的方法， 4 //尽最大努力去减少实现这个“顺序存取”的特性的数据存储(例如链表)的什么鬼，对于 5 //随机存取数据(例如数组)的类应该优先使用AbstractList 6 //从上面就可以大概知道，AbstractSwquentialList这个类是为了减少LinkedList这种顺//序存取的类的代码复杂度而抽象的一个类， 7 This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a "sequential access" data store (such as a linked list). For random access data (such as an array), AbstractList should be used in preference to this class. 8 9 //这一段大概讲的就是这个AbstractSequentialList这个类和AbstractList这个类是完全//相反的。比如get、add这个方法的实现10 This class is the opposite of the AbstractList class in the sense that it implements the "random access" methods (get(int index), set(int index, E element), add(int index, E element) and remove(int index)) on top of the list's list iterator, instead of the other way around.11 12 //这里就是讲一些我们自己要继承该类，该做些什么事情，一些规范。13 To implement a list the programmer needs only to extend this class and provide implementations for the listIterator and size methods. For an unmodifiable list, the programmer need only implement the list iterator's hasNext, next, hasPrevious, previous and index methods.14 15 For a modifiable list the programmer should additionally implement the list iterator's set method. For a variable-size list the programmer should additionally implement the list iterator's remove and add methods.16 17 The programmer should generally provide a void (no argument) and collection constructor, as per the recommendation in the Collection interface specification.

[IMG_262](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　　　实现的接口



　　　　　　　　1、List接口：列表，add、set、等一些对列表进行操作的方法

　　　　　　　　2、Deque接口：有队列的各种特性，

　　　　　　　　3、Cloneable接口：能够复制，使用那个copy方法。

　　　　　　　　4、Serializable接口：能够序列化。

　　　　　　　　5、应该注意到没有RandomAccess：那么就推荐使用iterator，在其中就有一个foreach，增强的for循环，其中原理也就是iterator，我们在使用的时候，使用foreach或者iterator都可以。

三、构造方法

　　　　　　两个构造方法(两个构造方法都是规范规定需要写的。看了AbstractSequentialList的api应该会知道)

　　　　　　　　1、空的构造方法

1 /\*\*2 \* Constructs an empty list.3 \*/4 public LinkedList() {5 }

　　　　　　　　2、有参构造

[IMG_264](http://www.cnblogs.com/whgk/p/javascript:void(0);)

1 /\*\* 2 \* Constructs a list containing the elements of the specified 3 \* collection, in the order they are returned by the collection's 4 \* iterator. 5 \* 6 \* @param c the collection whose elements are to be placed into this list 7 \* @throws NullPointerException if the specified collection is null 8 \*/ 9 //将集合c中的各个元素构建成LinkedList链表。10 public LinkedList(Collection<? extends E> c) {11 this();12 addAll(c);13 }

[IMG_265](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　　　　　addAll(c);

public boolean addAll(Collection<? extends E> c) {//继续往下看

return addAll(size, c);

}

　　　　　　　　　　addAll(size，c)：这个方法，能包含三种情况下的添加，我们这里分析的只是构造方法，空链表的情况(情况一)看的时候只需要按照不同的情况分析下去就行了。

通过这里的源码分析，就可以得知linkedList是一个双向链表，而不是双向循环链表。

[IMG_266](http://www.cnblogs.com/whgk/p/javascript:void(0);)

//真正核心的地方就是这里了，记得我们传过来的是size，c

public boolean addAll(int index, Collection<? extends E> c) {//检查index这个是否为合理。这个很简单，自己点进去看下就明白了。 checkPositionIndex(index);//将集合c转换为Object数组 a

Object[] a = c.toArray();//数组a的长度numNew，也就是由多少个元素

int numNew = a.length;

if (numNew == 0)//集合c是个空的，直接返回false，什么也不做。

return false;//集合c是非空的，定义两个节点(内部类)，每个节点都有三个属性，item、next、prev。注意：不要管这两个什么含义，就是用来做临时存储节点的。这个Node看下面一步的源码分析，Node就是linkedList的最核心的实现，可以直接先跳下一个去看Node的分析

Node<E> pred, succ;//构造方法中传过来的就是index==size

if (index == size) {//linkedList中三个属性：size、first、last。 size：链表中的元素个数。 first：头节点 last：尾节点，就两种情况能进来这里  
  
//情况一、：构造方法创建的一个空的链表，那么size=0，last、和first都为null。linkedList中是空的。什么节点都没有。succ=null、pred=last=null  
  
//情况二、：链表中有节点，size就不是为0，first和last都分别指向第一个节点，和最后一个节点，在最后一个节点之后追加元素，就得记录一下最后一个节点是什么，所以把last保存到pred临时节点中。

succ = null;

pred = last;

} else {//情况三、index！=size，说明不是前面两种情况，而是在链表中间插入元素，那么就得知道index上的节点是谁，保存到succ临时节点中，然后将succ的前一个节点保存到pred中，这样保存了这两个节点，就能够准确的插入节点了  
 //举个简单的例子，有2个位置，1、2、如果想插数据到第二个位置，双向链表中，就需要知道第一个位置是谁，原位置也就是第二个位置上是谁，然后才能将自己插到第二个位置上。如果这里还不明白，先看一下文章开头对于各种链表的删除，add操作是怎么实现的。

succ = node(index);

pred = succ.prev;

}//前面的准备工作做完了，将遍历数组a中的元素，封装为一个个节点。

for (Object o : a) {

@SuppressWarnings("unchecked") E e = (E) o;  
//pred就是之前所构建好的，可能为null、也可能不为null，为null的话就是属于情况一、不为null则可能是情况二、或者情况三

Node<E> newNode = new Node<>(pred, e, null);//如果pred==null，说明是情况一，构造方法，是刚创建的一个空链表，此时的newNode就当作第一个节点，所以把newNode给first头节点

if (pred == null)

first = newNode;

else//如果pred！=null，说明可能是情况2或者情况3，如果是情况2，pred就是last，那么在最后一个节点之后追加到newNode，如果是情况3，在中间插入，pred为原index节点之前的一个节点，将它的next指向插入的节点，也是对的

pred.next = newNode;

//然后将pred换成newNode，注意，这个不在else之中，请看清楚了。  
 pred = newNode;

}

if (succ == null) {//如果succ==null，说明是情况一或者情况二，  
情况一、构造方法，也就是刚创建的一个空链表，pred已经是newNode了，last=newNode，所以linkedList的first、last都指向第一个节点。  
情况二、在最后节后之后追加节点，那么原先的last就应该指向现在的最后一个节点了，就是newNode。

last = pred;

} else {//如果succ！=null，说明可能是情况三、在中间插入节点，举例说明这几个参数的意义，有1、2两个节点，现在想在第二个位置插入节点newNode，根据前面的代码，pred=newNode，succ=2，并且1.next=newNode，  
1已经构建好了，pred.next=succ，相当于在newNode.next = 2； succ.prev = pred，相当于 2.prev = newNode， 这样一来，这种指向关系就完成了。first和last不用变，因为头节点和尾节点没变

pred.next = succ;//。。

succ.prev = pred;

}

//增加了几个元素，就把 size = size +numNew 就可以了

size += numNew;

modCount++;

return true;

}

[IMG_267](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　　　　　Node：内部类

　　　　 item:当前节点的值

 　　　　 prev(previous)：指向当前节点的前一个节点

 　　 　　next：指向当前节点的后一个节点

[IMG_268](http://www.cnblogs.com/whgk/p/javascript:void(0);)

//根据前面介绍双向链表就知道这个代表什么了，linkedList的奥秘就在这里。

private static class Node<E> {

E item;

Node<E> next;

Node<E> prev;

Node(Node<E> prev, E element, Node<E> next) {

this.item = element;

this.next = next;

this.prev = prev;

}

}

[IMG_269](http://www.cnblogs.com/whgk/p/javascript:void(0);)

三、常用方法

　　　　　　常规，先看增、删、查询方法吧

　　　　　　add(E)

[IMG_270](http://www.cnblogs.com/whgk/p/javascript:void(0);)

/\*\*

\* 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})

\*///看上面第一句话的注释，在这个链表的末尾追加指定的元素

public boolean add(E e) {

//在末尾追加元素的方法。 linkLast(e);

return true;

}

[IMG_271](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　　　linkLast(e)

[IMG_272](http://www.cnblogs.com/whgk/p/javascript:void(0);)

/\*\*

\* Links e as last element.

\*/

void linkLast(E e) {

final Node<E> l = last; //临时节点l(L的小写)保存last，也就是l指向了最后一个节点

final Node<E> newNode = new Node<>(l, e, null);//将e封装为节点，并且e.prev指向了最后一个节点

last = newNode;//newNode成为了最后一个节点，所以last指向了它

if (l == null) //判断是不是一开始链表中就什么都没有，如果没有，则newNode就成为了第一个节点，first和last都要指向它

first = newNode;

else //正常的在最后一个节点后追加，那么原先的最后一个节点的next就要指向现在真正的最后一个节点，原先的最后一个节点就变成了倒数第二个节点

l.next = newNode;

size++;//添加一个节点，size自增

modCount++;

}

[IMG_273](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　remove(Object o)

[IMG_274](http://www.cnblogs.com/whgk/p/javascript:void(0);)

/\*\*

\* Removes the first occurrence of the specified element from this list,

\* if it is present. If this list does not contain the element, it is

\* unchanged. More formally, removes the element with the lowest index

\* {@code i} such that

\* <tt>(o==null&nbsp;?&nbsp;get(i)==null&nbsp;:&nbsp;o.equals(get(i)))</tt>

\* (if such an element exists). Returns {@code true} if this list

\* contained the specified element (or equivalently, if this list

\* changed as a result of the call).

\*

\* @param o element to be removed from this list, if present

\* @return {@code true} if this list contained the specified element

\*///首先通过看上面的注释，我们可以知道，如果我们要移除的值在链表中存在多个一样的值，那么我们会移除index最小的那个，也就是最先找到的那个值，如果不存在这个值，那么什么也不做

public boolean remove(Object o) {//这里可以看到，linkedList也能存储null

if (o == null) {//循环遍历链表，直到找到null值，然后使用unlink移除该值。下面的这个else中也一样

for (Node<E> x = first; x != null; x = x.next) {

if (x.item == null) {

unlink(x);

return true;

}

}

} else {

for (Node<E> x = first; x != null; x = x.next) {

if (o.equals(x.item)) {

unlink(x);

return true;

}

}

}

return false;

}

[IMG_275](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　　　unlink(x)

[IMG_276](http://www.cnblogs.com/whgk/p/javascript:void(0);)

/\*\*

\* Unlinks non-null node x.

\*///不能传一个null值过，注意，看之前要注意之前的next、prev这些都是谁。

E unlink(Node<E> x) {

// assert x != null;//拿到节点x的三个属性

final E element = x.item;

final Node<E> next = x.next;

final Node<E> prev = x.prev;

//这里开始往下就进行移除该元素之后的操作，也就是把指向哪个节点搞定。

if (prev == null) {//说明移除的节点是头节点，则first头节点应该指向下一个节点

first = next;

} else {//不是头节点，prev.next=next：有1、2、3，将1.next指向3

prev.next = next;//然后解除x节点的前指向。

x.prev = null;

}

if (next == null) {//说明移除的节点是尾节点

last = prev;

} else {//不是尾节点，有1、2、3，将3.prev指向1. 然后将2.next=解除指向。

next.prev = prev;

x.next = null;

}//x的前后指向都为null了，也把item为null，让gc回收它

x.item = null;

size--; //移除一个节点，size自减

modCount++;

return element; //由于一开始已经保存了x的值到element，所以返回。

}

[IMG_277](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　get(index)查询元素的方法

[IMG_278](http://www.cnblogs.com/whgk/p/javascript:void(0);)

/\*\*

\* 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}

\*///这里没有什么，重点还是在node(index)中

public E get(int index) {

checkElementIndex(index);

return node(index).item;

}

[IMG_279](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　node(index)

[IMG_280](http://www.cnblogs.com/whgk/p/javascript:void(0);)

/\*\*

\* Returns the (non-null) Node at the specified element index.

\*///这里查询使用的是先从中间分一半查找

Node<E> node(int index) {

// assert isElementIndex(index);//"<<":\*2的几次方 “>>”:/2的几次方，例如：size<<1：size\*2的1次方，//这个if中就是查询前半部分

if (index < (size >> 1)) {//index<size/2

Node<E> x = first;

for (int i = 0; i < index; i++)

x = x.next;

return x;

} else {//前半部分没找到，所以找后半部分

Node<E> x = last;

for (int i = size - 1; i > index; i--)

x = x.prev;

return x;

}

}

[IMG_281](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　indexOf(Object o)

[IMG_282](http://www.cnblogs.com/whgk/p/javascript:void(0);)

//这个很简单，就是通过实体元素来查找到该元素在链表中的位置。跟remove中的代码类似，只是返回类型不一样。

public int indexOf(Object o) {

int index = 0;

if (o == null) {

for (Node<E> x = first; x != null; x = x.next) {

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;

}

[IMG_283](http://www.cnblogs.com/whgk/p/javascript:void(0);)

　　　　　　小小总结一下。

　　　　　　　　如果你搞懂了这几个方法，那么其他的方法也就差不多了。可以自己尝试去看看，看了上面几个方法之后，应该有以下几点收获

　　　　　　　　　　1、linkedList：双向链表

　　　　　　　　　　2、能存放null值

　　　　　　　　　　3、在进行增加、删除时，虽然要先遍历到指定位置，但是不需要跟移动大量数据，很轻松就完成了任务。

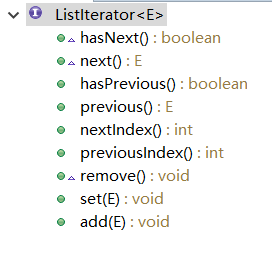
　　　　　　　　　　4、数据结构很重要。这是我看源码后的第一感觉。

四、ListIterator的使用，LinkedList特殊的东西

　　　　　　　　在LinkedList中除了有一个Node的内部类外，应该还能看到另外两个内部类，那就是ListItr，还有一个是DescendingIterator。

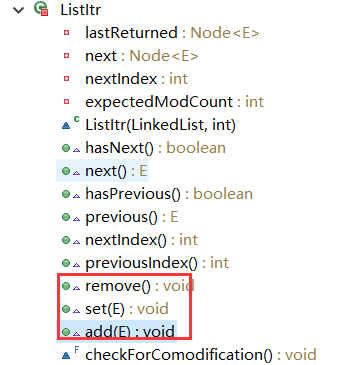
　　　　ListItr内部类

　　　　　　看一下他的继承结构，发现只继承了一个ListIterator，到ListIterator中一看，



　　　　　　看到方法名之后，就发现不止有向后迭代的方法，还有向前迭代的方法，所以我们就知道了这个ListItr这个内部类干嘛用的了，就是能让linkedList不光能像后迭代，也能向前迭代，其中源码非常简单，这里就不一一讲解了

　　　　　　　看一下ListItr中的方法，可以发现，在迭代的过程中，还能移除、修改、添加值得操作。



　　　　DescendingIterator内部类

[IMG_286](http://www.cnblogs.com/whgk/p/javascript:void(0);)

/\*\*

\* Adapter to provide descending iterators via ListItr.previous

\*/

看一下这个类，还是调用的ListItr，作用是封装一下Itr中几个方法，让使用者以正常的思维去写代码，例如，在从后往前遍历的时候，也是跟从前往后遍历一样，使用next等操作，而不用使用特殊的previous。

private class DescendingIterator implements Iterator<E> {

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

public boolean hasNext() {

return itr.hasPrevious();

}

public E next() {

return itr.previous();

}

public void remove() {

itr.remove();

}

}

[IMG_287](http://www.cnblogs.com/whgk/p/javascript:void(0);)

五、总结

　　　　　　通过查看LinkedList源码，我们知道以下几点

　　　　　　1、linkedList本质上是一个双向链表，通过一个Node内部类实现的这种链表结构。

　　　　　　2、能存储null值

　　　　　　3、跟arrayList相比较，就真正的知道了，LinkedList在删除和增加等操作上性能好，而ArrayList在查询的性能上号

　　　　　　4、从源码中看，它不存在容量不足的情况

　　　　　　5、linkedList不光能够向前迭代，还能像后迭代，并且在迭代的过程中，可以修改值、添加值、还能移除值。

　　　　　　6、linkedList不光能当链表，还能当队列使用，具体的可以自己去先看看其中的奥秘，这个我就归结到以后写Deque时在总结吧。