**Java JDK1.7的LinkedList**

@author ixenos

**LinkedList**

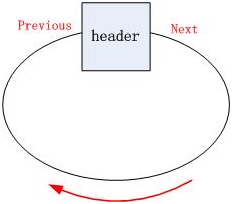
　　LinkedList是List接口的双向链表实现，JDK1.7以前是双向循环链表，以后是双向非循环链表；

　　由于是链表结构，所以长度没有限制；而且添加/删除元素的时候，只需要改变指针的指向（把链表断开，插入/删除元素，再把链表连起来）即可，非常方便，而ArrayList却需要重整数组 (add/remove中间元素)。所以LinkedList适合用于添加/删除操作频繁的情况

**JDK1.7以前 双向循环链表**

 　　在JDK 1.7之前（此处使用JDK1.6来举例），LinkedList是通过headerEntry实现的一个循环链表的：

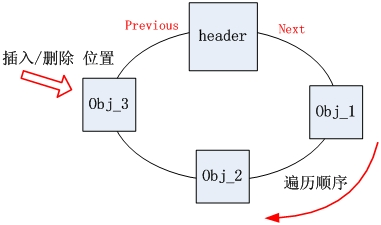
　　1.先初始化一个空的Entry，用来做header，然后首尾相连，形成一个循环链表：



1 privatetransient Entry<E>header =new Entry<E>(null,null,null);

2 public LinkedList() {header.next =header.previous =header; } //构造方法先生成一个头结点

        2.每次添加/删除元素都是默认在链尾操作：



　　对应此处，就是在header前面操作，因为遍历是next方向的，所以在header前面操作，就相当于在链表尾操作。

　　如下面的插入操作addBefore以及图示，如果插入obj\_3，只需要修改header.previous和obj\_2.next指向obj\_3即可：

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

[复制代码](javascript:void(0);)

1 private Entry<E> addBefore(Eo) { //取自JDK1.6-LinkedList，有删改

2

3 Entry<E>newEntry = new Entry<E>(o,header,header.previous);

4

5 newEntry.previous.next = newEntry;

6

7 newEntry.next.previous = newEntry;

8

9 size++;

10

11 modCount++;

12

13 return newEntry;

14

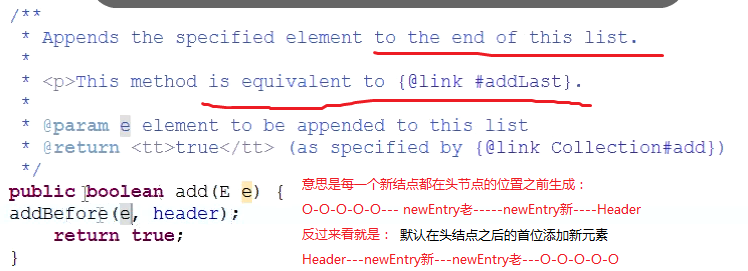
15 }

16

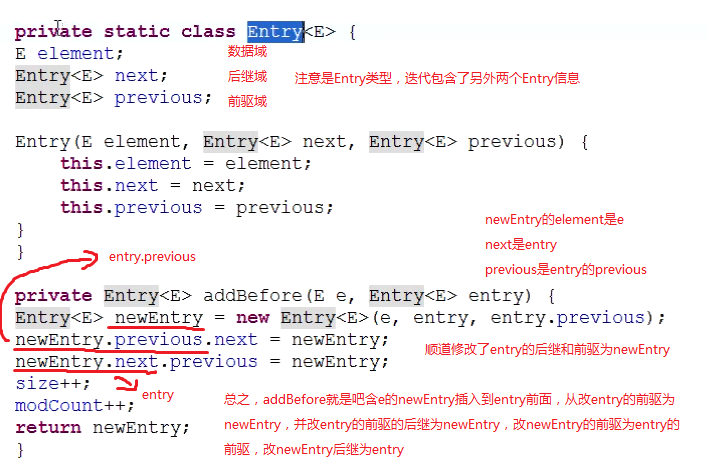
17

[复制代码](javascript:void(0);)

* *下面是我之前研究LinkedList源码时做的图片笔记，与以上内容重复，可略过：*

**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**JDK1.7 双向非循环链表**

　　在JDK 1.7，1.6的headerEntry循环链表被替换成了firstEntry和lastEntry组成的非循环链表。

　　在初始化的时候，不用去new一个Entry：

1 public LinkedList() { }

　　而是首尾两个Node对象(first、last)放在域中等待实例化，然后用构造器构造出来，而且一开始first的Next指向last，last的Previous指向first（其他null）：



　　在插入/删除的时候，也是默认在链尾操作。

　　把插入的obj当成newLast，挂在oldLast的后面，另外还要先判断first是否为空，如果为空则first = obj。

　　如下面的插入方法linkLast，在尾部操作，只需要把obj\_3.next指向obj\_4即可：

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

[复制代码](javascript:void(0);)

1 void linkLast(E e) { //取自JDK1.7-LinkedList，有删改

2

3 final Node<E> l = last;

4

5 final Node<E> newNode =new Node<>(l, e,null);

6

7 last = newNode;

8

9 if (l ==null)

10

11 first = newNode;

12

13 else

14

15 l.next = newNode;

16

17 size++;

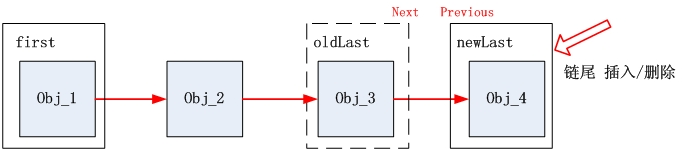
18

19 modCount++;

20

21 }

[复制代码](javascript:void(0);)



**对比总结**

 　　JDK 1.7中的first/last对比以前的header有下面几个好处：

　　1.first / last有更清晰的链头、链尾概念，代码看起来更容易明白。

　　2.first / last方式能节省new一个headerEntry。（实例化headerEntry是为了让后面的方法更加统一，否则会多很多header的空校验）

　　3.在链头/尾进行插入/删除操作，first /last方式更加快捷：

　　　　插入/删除操作按照位置，分为两种情况：

**在中间**插入/删除，两者都是一样，先遍历找到index，然后修改链表index处两头的指针

**在两头**，对于循环链表来说，由于首尾相连，还是需要处理两头的指针。而非循环链表只需要处理一边first.previous/last.next，所以理论上非循环链表更高效

　　4.对于遍历来说，两者都是链表指针循环，所以遍历效率是一样的