[JDK1.7-LinkedList循环链表优化](http://blog.csdn.net/tiwerbao/article/details/8227689)

标签： [JDK1.7](http://www.csdn.net/tag/JDK1.7)[LinkedList](http://www.csdn.net/tag/LinkedList)[循环链表](http://www.csdn.net/tag/%e5%be%aa%e7%8e%af%e9%93%be%e8%a1%a8)

2012-11-26 21:48 1543人阅读 [评论](http://blog.csdn.net/tiwerbao/article/details/8227689#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/tiwerbao/article/details/8227689#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

Java相关（30） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg JDK 源码（14） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

         最近在看jdk1.7的时候，发现了一个LinkedList的改良，baidu了一把，发现还没有人讨论这个问题。所以我自己思考了一下，在这里和大家分享，讨论一把！欢迎大家拍砖，讨论起来，把这个问题搞明白 : )

         首先，简单介绍一下LinkedList：

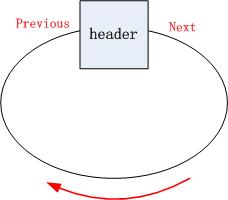
LinkedList是List接口的双向链表实现。由于是链表结构，所以长度没有限制；而且添加/删除元素的时候，只需要改变指针的指向（把链表断开，插入/删除元素，再把链表连起来）即可，非常方便，而ArrayList却需要重整数组 (add/remove中间元素)。所以LinkedList适合用于添加/删除操作频繁的情况。

--------------------------------Let’s go tothe code --------------------------------

         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。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：

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

**public** LinkedList() {header.next =header.previous =header; }



        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。

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

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

    newEntry.previous.next = newEntry;

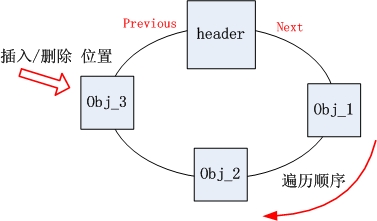
    newEntry.next.previous = newEntry;

    size++;

    modCount++;

**return** newEntry;

}



---------------------------------------------

         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。

**public** LinkedList() { }



         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。

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

**final** Node<E> l = last;

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

        last = newNode;

**if** (l ==**null**)

            first = newNode;

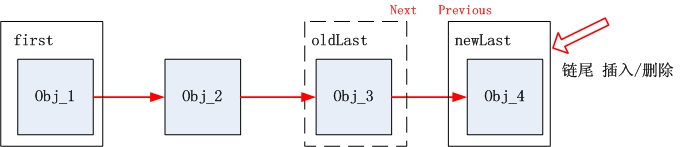
**else**

            l.next = newNode;

        size++;

        modCount++;

}



---------------------------------------------

**To sum up: 【1.6-header循环链表】 V.S 【1.7-first/last非循环链表】**

         在我看来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：中间 和 两头。

**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。恰恰在两头(链头/链尾) 操作是最普遍的】

（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。）

        上面三点就是本人的小结，由于都是小改良，所以目测 1.7-LinkedList和1.6的对比，效率上应该还是差不多（没有试验）。也许也是因为这点，所以没有人拿出来讨论。