# LinkedList(jdk1.6.0\_13)

LinkedList是用双向循环链表实现的，下面看一下具体的实现方式。

# 属性



链表的头节点。头节点的作用是当插入（删除）的节点为第一个节点时，能够和插入（删除）非第一个节点的操作相同。

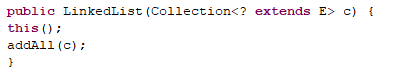


链表长度。

# 构造器

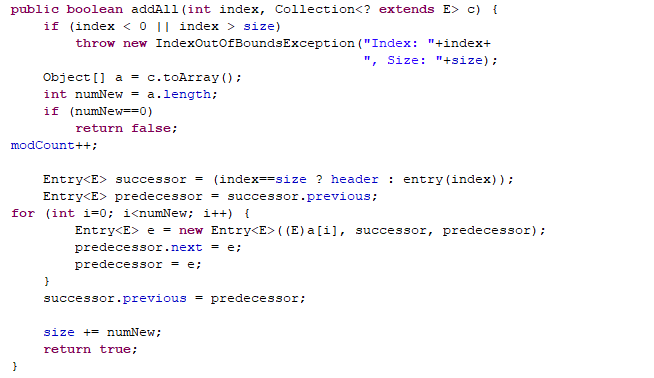


初始化了链表。

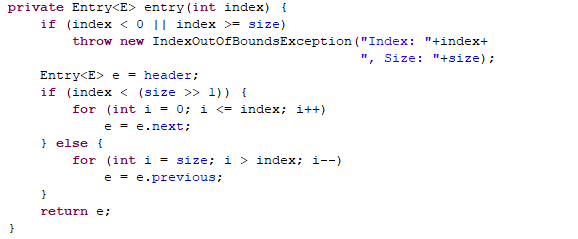


将一个已有集合的所有元素添加到链表中。看一下addAll方法：





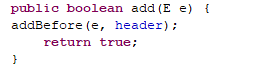
index参数表示插入的位置。如果index<0或者index>size，抛出越界异常。这里是在链表的尾部插入节点，首先获取插入节点的后继节点successor，前驱节点predecessor，然后遍历每个要插入的节点，将插入节点链接到successor与predecessor之间，每次插入重置插入节点的前驱节点predecessor。下面看一下entry方法：



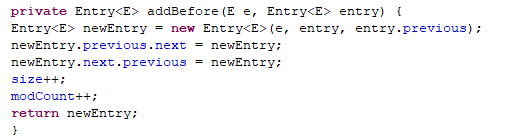
先判读index是在链表的前半部分还是后半部分，如果在前半部分则从前往后遍历查找节点，如果在后半部分则从后往前遍历查找元素。

# 基本方法

## 3.1add



在链表尾部插入元素。



很基本的双向链表的插入操作，这里记entry为后继节点，entry.previous为前驱节点。插入过程如下：创建新节点，将next指向后继节点，将previous指向前驱。将后继节点的previous指向新节点，将前驱节点的next指向新节点。

## 3.2add(int, e)



在指定位置插入，如果index==size，则在链表尾部插入，否则调用entry()查找要插入节点的位置的元素。

## 3.3addFirst



在链表头插入。

## 3.4addLast



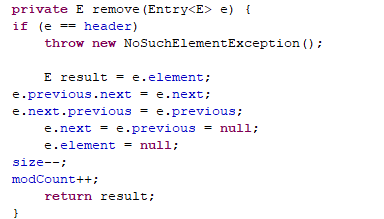
在链表尾部插入。

## 3.5remove



删除第一个元素。





基本的双向链表删除操作，将删除元素的前驱的next指向删除元素的后继，将删除元素的后继的previous指向删除元素的前驱，然后将删除元素的next，previous，element置null。

## 3.6其他删除操作



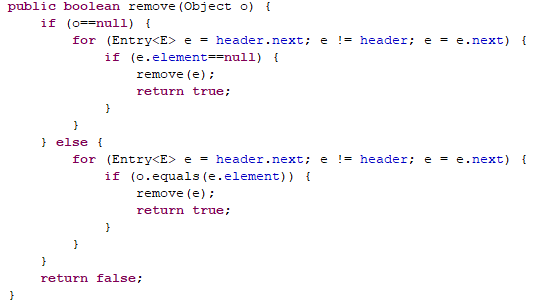
删除指定位置节点。



删除头部节点。



删除尾部节点。

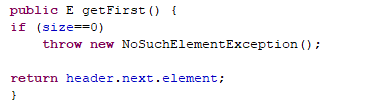


删除指定值的元素。

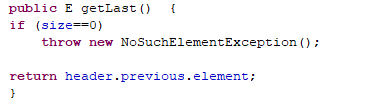
## 3.7get



获取指定位置的元素。



获取第一个元素。



获取最后一个元素。

# LinkedList(jdk1.7.0\_67)

LinkedList在jdk1.7中的实现有些许差别。在jdk1.6中使用的双向循环链表，使用了一个头节点来维护链表的插入删除操作。而在jdk1.7中使用的是双向链表，并没有添加额外的节点。下面具体来看一下。

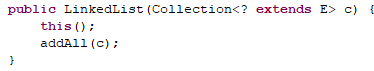


链表第一个节点。

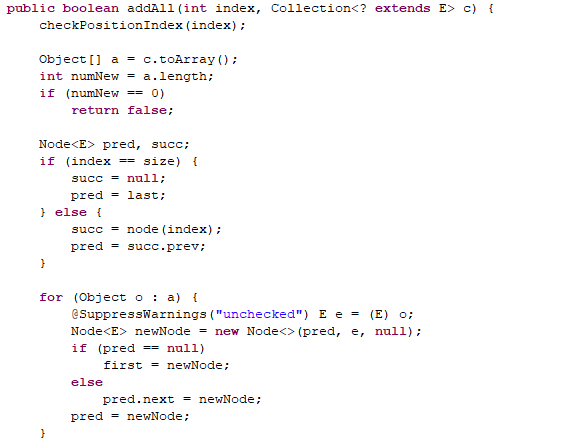


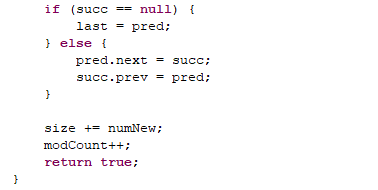
链表尾节点。

# 构造器





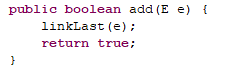




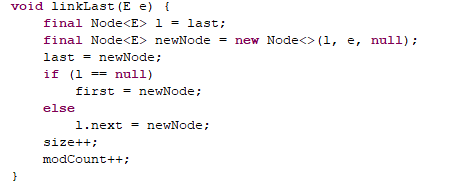
这里将jdk1.6中检查越界的代码封装成了checkPositionIndex方法。下面的就跟jdk1.6中的操作差不多了，首先查找插入节点的前驱pred和后继节点succ，然后遍历插入节点，将元素链接到pred与succ之间。

# 基本方法

## 5.1add

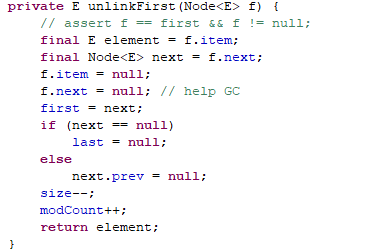
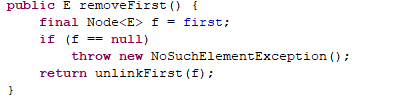


这里使用了linkLast替换了jdk1.6中的addBefore方法，具体看一下linkLast方法的实现：



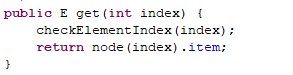
如果是第一次插入将last，first都指向新节点，如果不是第一次插入，先记录last，然后将last指向新节点，将以前的last的next指向新节点。

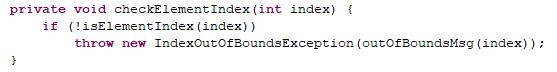
## 5.2删除remove



每次删除的都是first节点，首先将要要删除的节点的后继节点记录下来next，然后将first节点的next，item置null，将first指向next，如果next为null，将last置null；如果next不为null，将next的previous置null。

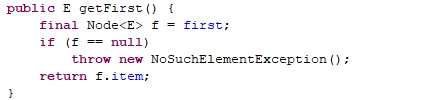
## 5.3get



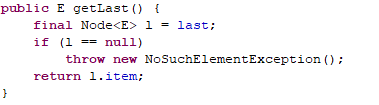




首先检查index是否越界，然后查找元素。



获取第一个元素。



获取最后一个元素的值。