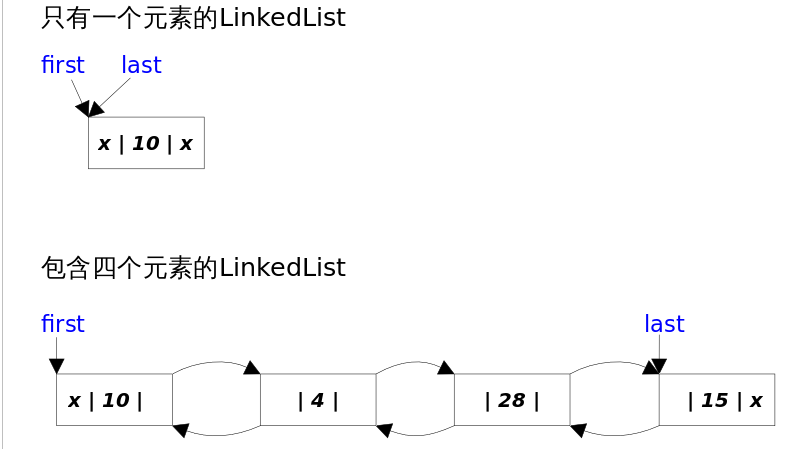
1. LinkedList基础
2. 总体介绍
3. LinkedList用途

LinkedList同时实现了List接口和Deque接口，也就是说它既可以看作一个顺序容器，又可以看作一个队列（Queue），同时又可以看作一个栈（Stack）。这样看来，LinkedList简直就是个全能冠军。当你需要使用栈或者队列时，可以考虑使用LinkedList，一方面是因为Java官方已经声明不建议使用Stack类，更遗憾的是，Java里根本没有一个叫做Queue的类（它是个接口名字）。关于栈或队列，现在的首选是ArrayDeque，它有着比LinkedList（当作栈或队列使用时）有着更好的性能。

1. LinkedList原理 -- 是双向链表而不是双向循环链表

LinkedList底层通过双向链表实现，主要是插入和删除元素时双向链表的维护过程，双向链表的每个节点用内部类Node表示。LinkedList通过first和last引用分别指向链表的第一个和最后一个元素。注意这里没有所谓的哑元，当链表为空的时候first和last都指向null。



1. LinkedList实现
   1. 构造方法
2. public LinkedList() {}
3. public LinkedList(Collection<? extends E> paramCollection)

{

this();

addAll(paramCollection);

}

* 1. Node节点类 -- 链表的关键实现

private static class Node<E>

{ //每个Node都有一个next，一个prev，一个当前元素item

//没有实例化，只能是一个属性并没有分配空间

E item;

Node<E> next;

Node<E> prev;

//构造方法

Node(Node<E> paramNode1, E paramE, Node<E> paramNode2)

{

item = paramE;

next = paramNode2;

prev = paramNode1;

}

}

说明：为追求效率LinkedList没有实现同步（synchronized），如果需要多个线程并发访问，可以先采用Collections.synchronizedList()方法对其进行包装。

1. 具体方法实例
2. 添加元素
   1. add()
3. public boolean add(E paramE)

public void addLast(E paramE)

说明：将指定的元素追加到此列表的末尾。

方法：linkLast()

void linkLast(E paramE)

{ //复制last对象引用，赋值到localNode1

Node localNode1 = last;

//将新增对象添加到localNode2，prev指向localNode1

Node localNode2 = new Node(localNode1, paramE, null);

//将localNode2对象引用复制到last上

last = localNode2;

//当localNode1为空时，localNode2对象引用赋值到first上，否则将赋值到next上

if (localNode1 == null) {

first = localNode2;

} else {

next = localNode2;//此时localNode1对象引用取代了原来的last

} //last对象引用取代了new出来的localNode2

size += 1;

modCount += 1;

}

addFirst()方法和这个类似，将元素添加到链表的头部

1. public void add(int paramInt, E paramE)

说明：在此列表中的指定位置插入指定的元素。 将当前位于该位置的元素（如果有）和任何后续元素（向其索引添加一个）移动一位。1.先根据index找到要插入的位置；2.修改引用，完成插入操作。

1. public void add(int paramInt, E paramE)

{ // 越界检测

checkPositionIndex(paramInt);

if (paramInt == size) { //当index为最大限度时，插入到最后一个位置

linkLast(paramE);

} else {

linkBefore(paramE, node(paramInt));

}

}

1. void linkBefore(E paramE, Node<E> paramNode)

{

Node localNode1 = prev; //将前一元素对象引用赋值给localNode1

//将localNode1赋值给localNode2的prev，将localNode2的next赋值给取出元素

Node localNode2 = new Node(localNode1, paramE, paramNode);

prev = localNode2;//将localNode2赋值给本元素的prev

if (localNode1 == null) {

first = localNode2;

} else {

//将localNode1的next赋值给localNode2

next = localNode2;

}

size += 1;

modCount += 1;

}

1. Node<E> node(int paramInt)

{ //size>>1指除2，>>运算符比<运算符优先级高

if (paramInt < size >> 1)

{ //从前往后查找

localNode = first;

for (i = 0; i < paramInt; i++) {

localNode = next;

}

return localNode;

} //从后往前查找

Node localNode = last;

for (int i = size - 1; i > paramInt; i--) {

localNode = prev;

}

return localNode;

}

* 1. set() 方法

说明：用指定的元素替换此列表中指定位置的元素，返回被替换的元素。

public E set(int paramInt, E paramE)

{

checkElementIndex(paramInt); //越界检测

Node localNode = node(paramInt);// 取出对应位置的node对象

Object localObject = item; //将旧对象的item保存

item = paramE; //将传入的paramE赋值给该对象的item

return (E)localObject; //返回被替换的item

}

* 1. offer() -- 底层调用add()相关方法

offer()属于queue队列方法，add()属于list方法

1. public boolean offer(E paramE)

将指定的元素添加为此列表的尾部（最后一个元素）。

1. public boolean offerFirst(E paramE)

在此列表的前面插入指定的元素。

1. public boolean offerLast(E paramE)

在该列表的末尾插入指定的元素。

1. 删除元素
   1. remove()
2. public boolean remove(Object paramObject)

从列表中删除指定元素的第一个出现（如果存在）。 如果此列表不包含该元素，则它将保持不变。 更正式地，删除具有最低索引i的元素

方法：E unlink(Node<E> paramNode)

{ //将该node的item，prev，next复制出来

Object localObject = item;

Node localNode1 = next;

Node localNode2 = prev;

if (localNode2 == null) //如果localNode2为null，将localNode1赋值给first

{

first = localNode1;

}

else //如果localNode2不为null，将localNode1赋值给localNode2的next

{

next = localNode1;

prev = null; //将删除对象的prev设为null

}

if (localNode1 == null) //如果localNode1为null，将localNode2赋值给last

{

last = localNode2;

}

else //如果localNode1不为null，将localNode2赋值给localNode1的prev

{

prev = localNode2;

next = null;//将删除对象的next设为null

}

item = null;

size -= 1;

modCount += 1;

return (E)localObject;

}

1. public E remove(int paramInt)

{

checkElementIndex(paramInt); //边界检测

return (E)unlink(node(paramInt)); //先用node()获得指定元素，再调用unlink删除元素

}

1. remove() -- 默认调用removeFirst()，删除成功返回true

removeFirst() -- 从此列表中删除并返回第一个元素。

removeLast() -- 从此列表中删除并返回最后一个元素。

实例：

private E unlinkFirst(Node<E> paramNode)

{

Object localObject = item;

Node localNode = next;

item = null;

next = null; //将删除元素的next设为null

first = localNode;

if (localNode == null) {

last = null; //当first的next为null时，将last设为null

} else {

prev = null;//将localNode的prev设为null

}

size -= 1; //size减1

modCount += 1;

return (E)localObject;

}

* 1. poll() -- 底层调用unlink()方法，和remove()一样

1. public E poll()

检索并删除此列表的头（第一个元素）。

1. public E pollFirst()

检索并删除此列表的第一个元素，如果此列表为空，则返回 null 。

1. public E pollLast()

检索并删除此列表的最后一个元素，如果此列表为空，则返回 null 。

1. 获取元素
   1. get()
2. public E get(int paramInt) - 返回此列表中指定位置的元素。

{

checkElementIndex(paramInt); //边界检测

return (E)nodeitem; //node()获取指定位置的元素

}

1. public E getFirst() -- 返回此列表中的第一个元素。
2. public E getLast() -- 返回此列表中的最后一个元素。
   1. peek()
3. public E peek()

检索但不删除此列表的头（第一个元素）。

1. public E peekFirst()

检索但不删除此列表的第一个元素，如果此列表为空，则返回 null 。

1. public E peekLast()

检索但不删除此列表的最后一个元素，如果此列表为空，则返回 null 。

1. 获取index地址
2. public int indexOf(Object paramObject)

返回此列表中指定元素的第一次出现的索引，如果此列表不包含元素，则返回-1。

1. public int lastIndexOf(Object paramObject)

返回此列表中指定元素的最后一次出现的索引，如果此列表不包含元素，则返回-1。

1. 其他方法
2. public void clear()

从列表中删除所有元素，全部设为null。

1. public Object clone()

克隆一个新对象，返回为Object。修改新克隆数组不会改变原数组。

6. 堆栈方法

(1) public E pop() -- 先进后出

删除第一个元素，相当于从堆栈中弹出一个元素。

(2) public void push(E paramE)

添加到第一位，相当往堆栈中增加一个元素。

7. 队列方法 -- 先进先出

(1) offer() -- 往队尾添加一个元素

(2) poll() -- 删除队首第一个元素

(3) peek() -- 取出队首第一个元素

1. 总结
2. LinkedList内部是用双向链表实现的，维护了长度、头节点和尾节点，这决定了它有如下特点：
3. 按需分配空间，不需要预先分配很多空间
4. 不可以随机访问，按照索引位置访问效率比较低，必须从头或尾顺着链接找，效率为O(N/2)。
5. 不管列表是否已排序，只要是按照内容查找元素，效率都比较低，必须逐个比较，效率为O(N)。
6. 在两端添加、删除元素的效率很高，为O(1)。
7. 在中间插入、删除元素，要先定位，效率比较低，为O(N)，但修改本身的效率很高，效率为O(1)。
8. 如果列表长度未知，添加、删除操作比较多，尤其经常从两端进行操作，而按照索引位置访问相对比较少，则LinkedList就是比较理想的选择。