### 3.3 链表

* 1.什么是单链表 & 双向链表 & 循环链表 & 双向循环链表 & 静态链表。

[单链表](https://www.cnblogs.com/sun-haiyu/p/7267130.html)  
[双向链表](https://www.cnblogs.com/cmusketeer/p/9748719.html)  
[循环链表](http://www.cnblogs.com/sun-haiyu/archive/2017/08/01/7267510.html)  
[双向循环链表](https://www.cnblogs.com/lfalex0831/p/9674349.html)  
[静态链表](https://www.cnblogs.com/ityizhainan/p/6004964.html)

* 2.反转一个链表有哪些方式？

[点击查看答案](https://www.cnblogs.com/byrhuangqiang/p/4311336.html)

* 3.如何判断链表有环？

[点击查看答案](https://www.cnblogs.com/wuyuanyuan/p/8124657.html)

* 4.用Java语言设计一个LinkedList。
* public class LinkedListCopy<E> {  
    
   private Node<E> head = new Node<>(null);//整个单链表的头部  
   private int size =0;//单链表的长度  
    
   //结点类  
   private static class Node<E>{  
   public Node head = null;//结点的头部  
   public Node tail = null;//结点的尾部  
   E value;//结点中存放的值  
   public Node(E e){value = e;}  
   }  
    
   //迭代器类  
   private static class DieDaiQi<E>{  
    
   private LinkedListCopy<E> linkedListCopy;//模拟被迭代的LikedList  
   private Node<E> dangQian ;//存放处于焦点的结点  
    
   public DieDaiQi(LinkedListCopy<E> linkedListCopy){  
   this.linkedListCopy = linkedListCopy;  
   dangQian = linkedListCopy.head;//当你生成迭代器的时候，焦点开始在整个单链表的头部  
   }  
    
   //将当前的焦点移动一格  
   public boolean moveToNext(){  
   boolean falg = (dangQian.tail != null);  
   dangQian = dangQian.tail;  
   return falg;  
   }  
    
   //获取当前焦点对应的结点的值  
   public E next() {  
   E e = dangQian.value;  
   return e;  
   }  
    
   //重置迭代器  
   public void reSet(){  
   dangQian = linkedListCopy.head;  
   }  
    
   }  
    
    
   public LinkedListCopy(){  
    
   }  
    
   //获取当前单链表的迭代器  
   public DieDaiQi dieDaiQi(){  
   return new DieDaiQi(this);  
   }  
    
    
   //尾部添加元素  
   public void add(E e){  
   Node<E> dangQian = this.head.tail; //存放焦点对应的结点  
   Node<E> ago = head; //存放当前焦点前一个的结点  
   while(true) {  
   if(dangQian!=null){  
   //没有移动到尾部，将焦点移动一格  
   ago = dangQian;  
   dangQian = dangQian.tail;  
   }else {  
   //当移动道尾部的时候，就可以添加元素  
   dangQian = new Node<E>(e);  
   if(ago.tail == null){  
   ago.tail = dangQian;  
   dangQian.head = ago;  
   }  
   break;  
   }  
   }  
   size++;//添加元素，长度加1  
   }  
    
   //在指定位置插入元素  
   public void add(int i,E e){  
   if(i<0||i>=size()){  
   System.out.println("越界异常");  
   return;  
   }  
   if( i == this.size()){this.add(e);return;}//尾部插入  
   //插入添加  
   Node<E> node = new Node<>(e);  
   Node<E> dangQian = this.head.tail;//存放焦点对应的结点  
   Node<E> ago = this.head;//存放当前焦点前一个的结点  
   int j = 0;  
   while(dangQian != null)  
   {  
   if(i == j){  
    
   node.head = ago;  
   node.tail = dangQian;  
   dangQian.head = node;  
   ago.tail = node;  
   break;  
   }else{  
   j++;  
   ago = dangQian;  
   dangQian = dangQian.tail;  
   }  
   }  
   size++;//添加元素，长度加1  
   }  
    
    
   //删除元素  
   public void delete(int i){  
   if(i<0||i>=size()){  
   System.out.println("越界异常！");  
   return;  
   }  
   Node<E> dangQian = this.head.tail;  
   Node<E> ago = head;  
   int j=0;  
   while(dangQian != null){  
   if(j == i){  
   ago.tail = dangQian.tail;  
   dangQian.tail.head = ago;  
   break;  
   }else{  
   j++;  
   ago = dangQian;  
   dangQian = dangQian.tail;  
   }  
    
   }  
   size--;//删除元素长度减一  
   }  
    
   //修改元素  
   public void gaiValue(int i,E e){  
   if(i<0||i>=size()){  
   System.out.println("越界异常");  
   return;  
   }  
   Node<E> dangQian = this.head.tail;  
   Node<E> ago = null;  
   int j=0;  
   while(dangQian != null){  
   if(j == i){  
   dangQian.value = e;  
   break;  
   }else{  
   j++;  
   ago = dangQian;  
   dangQian = dangQian.tail;  
   }  
   }  
   }  
    
   //查询元素  
   public int serach(E e){  
    
   Node<E> dangQian = this.head.tail;  
   int j=0;  
   while(dangQian != null){  
    
   if(dangQian.value.equals(e)){  
   return j;  
   }else{  
   j++;  
   dangQian = dangQian.tail;  
   }  
    
   }  
    
   return -1;  
   }  
    
    
   //获取当前单链表的长度  
   public int size(){  
   return size;  
   }  
    
   //尾部添加另外一个单链表  
   public void addAll(LinkedListCopy<E> linkedListCopy){  
   Node<E> dangQian = linkedListCopy.head.tail;  
   while(dangQian != null){  
   this.add(dangQian.value);  
   dangQian = dangQian.tail;  
   }  
   }  
    
   //获取集合指定位置的元素值  
   public E get(int i){  
   if(i<0||i>=size()){  
   System.out.println("越界异常");  
   return null;  
   }  
   Node<E> dangQian = this.head.tail;  
   int j=0;  
   while(dangQian!=null){  
   if(j == i){  
   return dangQian.value;  
   }else{  
   j++;dangQian =dangQian.tail;  
   }  
   }  
   return null;  
   }  
    
   //还有其它的功能可以模拟，代码较多，笔者就不写了，理解上面的迭代器原理，什么都可以模拟的出来  
    
   public static void main(String[] args){  
    
   LinkedListCopy<String> linkedListCopy = new LinkedListCopy<String>();  
   DieDaiQi<String> dieDaiQi = linkedListCopy.dieDaiQi();//获取迭代器  
    
   //尾部增加  
   linkedListCopy.add("ss1");  
   linkedListCopy.add("ss2");  
   linkedListCopy.add("ss3");  
   linkedListCopy.add("ss4");  
    
   System.out.println("1\*\*\*\*\*尾部添加操作\*\*\*\*\*");  
   System.out.println("尾部添加操作后集合所有元素如下：");  
   while (dieDaiQi.moveToNext()){  
   System.out.println(dieDaiQi.next());  
   }  
    
   System.out.println("此时集合元素个数："+linkedListCopy.size());  
   System.out.println("1\*\*\*\*\*尾部添加操作\*\*\*\*\*\n");  
    
   //插入添加  
   linkedListCopy.add(0,"ss");  
   linkedListCopy.add(3,"ss2.5");  
    
   dieDaiQi.reSet();//重置迭代器  
   System.out.println("2\*\*\*\*\*插入添加操作\*\*\*\*\*");  
   System.out.println("插入添加操作后集合所有元素如下：");  
   while (dieDaiQi.moveToNext()){  
   System.out.println(dieDaiQi.next());  
   }  
   System.out.println("此时集合元素个数："+linkedListCopy.size());  
   System.out.println("2\*\*\*\*\*插入添加操作\*\*\*\*\*\n");  
    
   //删除  
   linkedListCopy.delete(3);  
   dieDaiQi.reSet();//重置迭代器  
   System.out.println("3\*\*\*\*\*删除操作\*\*\*\*\*");  
   System.out.println("删除操作后集合所有元素如下：");  
   while (dieDaiQi.moveToNext()){  
   System.out.println(dieDaiQi.next());  
   }  
   System.out.println("此时集合元素个数："+linkedListCopy.size());  
   System.out.println("3\*\*\*\*\*删除操作\*\*\*\*\*\n");  
    
   //修改  
   linkedListCopy.gaiValue(0,"ss0");  
   dieDaiQi.reSet();//重置迭代器  
   System.out.println("4\*\*\*\*\*修改操作\*\*\*\*\*");  
   System.out.println("修改操作后集合所有元素如下：");  
   while (dieDaiQi.moveToNext()){  
   System.out.println(dieDaiQi.next());  
   }  
   System.out.println("此时集合元素个数："+linkedListCopy.size());  
   System.out.println("4\*\*\*\*\*修改操作\*\*\*\*\*\n");  
    
    
   //查询操作  
   int serachInt = linkedListCopy.serach("ss8");  
   if(serachInt==-1) {  
   System.out.println("集合中不存在该元素！");  
   }else{  
   System.out.println("ss3存在的位置是：第" + (serachInt + 1) + "个");  
   }  
    
   System.out.println(linkedListCopy.get(serachInt));  
    
   }  
  }