目 录

第一章　顺序表

1.1　顺序表的使用

1.1.1　创建顺序表

```
//1.创建ArrayList实例
ArrayList<String> arrayList = new ArrayList<>();
//向上转型
List<String> arrayList = new ArrayList<>();
```

1.1.2　往顺序表中添加元素

add 一个参数

```
//2.添加元素
//add一个参数版本的方法是把元素添加到顺序表的末尾
arrayList.add("C");
arrayList.add("C++");
arrayList.add("Java");
arrayList.add("Python");
```

add 两个参数

```
//2.添加元素
//add两个参数版本的方法是把元素添加到顺序表的指定位置上
arrayList.add(2,"JavaScript");
//直接打印对象，会触发对象的toString方法
System.out.println(arrayList);
```

1.1.3　删除元素

```
//3.删除元素
//按位置删除
arrayList.remove(2);
//按照值删除，如果有多个相同的值，只会删除第一个
arrayList.remove("JavaScript");
//特殊情况，删除整型的
List<Integer> arrayList2 = new ArrayList<>();
for (int i = 0; i < 5; i++) {
    arrayList2.add(i + 1);
}
arrayList2.remove(2);//删除的是下标为2的元素而不是值为2的
arrayList2.remove(Integer.valueOf(2));//这个才是删除值为2的元素
```

1.1.4 查找

```java
//4.查找

//查找是否存在
boolean ret  = arrayList.contains("Java");
System.out.println("查找Java的结果: "+ ret);
//查找具体的位置
int index = arrayList.indexOf("Java");
System.out.println("查找Java的位置: "+ index);
```

1.1.5 获取/修改元素

```java
//5.获取元素/修改元素
//获取元素
arrayList.get(0);//获取下标为0的元素
//修改元素
arrayList.set(0,"C#");//把下标为0的元素修改为C#
```

1.1.6 迭代器：Iterator

迭代器是用来遍历集合的一种重要手段，在非线性结构迭代器会大显身手

```java
//6.遍历操作
//通过下标遍历
for (int i = 0; i < arrayList.size(); i++) {
    System.out.println(arrayList.get(i));
}

//通过迭代器来遍历
//先通过iterator方法获取到迭代器对象
Iterator<String> iterator = arrayList.iterator();
//再通过while循环来进行遍历
while (iterator.hasNext()){
    String elem = iterator.next();
    System.out.println(elem);
}

//使用for-each来遍历
for (String str:arrayList) {
    System.out.println(str);
}
```

1.1.7 练习：扑克牌游戏

```java
//一张扑克牌
public class Card {
```

```
    //花色
    protected String suit;
    //点数
    protected String rank;

    public Card(String suit, String rank) {
        this.suit = suit;
        this.rank = rank;
    }
    @Override
    public String toString(){
        return "(" + this.suit + this.rank + ")";
    }
}


import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class PokerName {
    //存放4种花色
    public static final String[] suits = {" "," "," "," "};
    //存放点数
    public static final String[] ranks = {"A","2","3","4","5","6","7","8","9","10"
        ,"J","Q","K"};

    private static List<Card> buyPoker(){
        List<Card> poker = new ArrayList<>();
        for (int i = 0; i < suits.length; i++) {
            for (int j = 0; j < ranks.length ; j++) {
                poker.add(new Card(suits[i],ranks[j]));
            }
        }
        poker.add(new Card("red","Joker"));
        poker.add(new Card("black","Joker"));
        return poker;
    }

    private static void shuffle(List<Card> poker){
        Random random = new Random();
        for (int i = poker.size()-1; i > 0 ; i--) {
            int pos  = random.nextInt(i);
            swap(poker,i,pos);
        }
    }

    private static void swap(List<Card> poker, int i, int pos) {
        Card tmp = poker.get(i);
        poker.set(i,poker.get(pos));
```

```
53          poker.set(pos,tmp);
54      }
55
56      public static void main(String[] args) {
57          //1.创建出一副扑克牌
58          List<Card> poker = buyPoker();
59          System.out.println(poker);
60          System.out.println("--------------------------------------");
61          //2.洗牌
62          shuffle(poker);
63          System.out.println(poker);
64          System.out.println("--------------------------------------");
65          shuffle(poker);
66          System.out.println(poker);
67          System.out.println("--------------------------------------");
68          //标准库自带的洗牌方法
69          Collections.shuffle(poker);
70          System.out.println(poker);
71          //3.发牌：每个玩家发5张牌，这五张牌用另一个ArrayList表示
72          List<List<Card>> players = new ArrayList<>();
73          Scanner scanner = new Scanner(System.in);
74          System.out.println("请输入玩家的个数：");
75          int num = scanner.nextInt();
76          for (int i = 0; i < num; i++) {
77              players.add(new ArrayList<Card>());
78          }
79          for (int i = 0; i < 5; i++) {
80              for (int j = 0; j < num; j++) {
81                  Card topCard = poker.remove(0);
82                  List<Card> player = players.get(j);
83                  player.add(topCard);
84              }
85          }
86          System.out.println(players);
87          System.out.println("--------------------------------------");
88          //4.展示手牌：
89          for (int i = 0; i < players.size(); i++) {
90              System.out.println("第" + (i + 1) + "位玩家的牌是：" + players.get(i))
                    ;
91          }
92      }
93  }
```

1.2 线性表的实现

1.2.1 空间管理

根据实际需求动态调整，并同时保证高效率。

发生上溢时，适当的扩大数组的容量，扩容策略：倍增式扩容。累计时间复杂度为 $O(n)$，分摊复杂度

为 $O(1)$。

```java
public class MyArrayList<E> {
    //属性
    private E[] data = null;
    //最大容量
    private int capacity = 4;
    //有效元素的个数
    private int size = 0;

    //构造方法
    public MyArrayList(){
        this.data = (E[])new Object[capacity];
    }

    //方法:增删改查

    //实现扩容
    private void realloc(){
        capacity <<= 1;
        E[] newData = (E[])new Object[capacity];
        for (int i = 0; i < size; i++) {
            newData[i] = data[i];
        }
        data = newData;
        newData = null;
    }

    //尾插
    public void add(E elem){
        if (size >= capacity){
            //如果满了，就需要扩容
            realloc();
        }
        data[size++] = elem;
    }
    //指定位置插入元素
    public void add(int index, E elem)throws IndexOutOfBoundsException{
        if(index < 0 || index > size){//等于size相当于是尾插
            throw new IndexOutOfBoundsException("下标越界");
        }
        if (size >= capacity){
            //如果满了，就需要扩容
            realloc();
        }
        for (int i = size - 1; i >= index; i--) {
            data[i + 1] = data[i];
        }
        data[index] = elem;
        size++;
    }
```

```
50
51      //按照下标位置删除元素
52      public E remove(int index) throws IndexOutOfBoundsException{
53          if (index < 0){
54              throw new IndexOutOfBoundsException("输入的下标小于下界（下界为0）,你
                    传入的下标是" + index + "。");
55          }
56          if (index >= size){
57              String str = "输入的下标超过上界（上界为" + (size - 1) + "）,你传入的
                    下标是" + index + "。";
58              throw new IndexOutOfBoundsException(str);
59          }
60          E elem = data[index];
61          for (int i = index; i < size - 1; i++) {
62              data[i] = data[i+1];
63          }
64          size--;
65          return elem;
66      }
67
68      //按照元素的值来删除元素
69      public boolean remove(E e){
70          int index = 0;
71          for (index = 0; index < size; index++) {
72              if (data[index].equals(e)){
73                  break;
74              }
75          }
76          if (index >= size){
77              return false;
78          }
79          for (int i = index; i < size - 1; i++) {
80              data[i] = data[i + 1];
81          }
82          size--;
83          return true;
84      }
85
86      //根据下标获取元素
87      public E get(int index)throws IndexOutOfBoundsException {
88          if (index < 0){
89              throw new IndexOutOfBoundsException("输入的下标小于下界（下界为0）,你
                    传入的下标是" + index + "。");
90          }
91          if (index >= size){
92              String str = "输入的下标超过上界（上界为" + (size - 1) + "）,你传入的
                    下标是" + index + "。";
93              throw new IndexOutOfBoundsException(str);
94          }
95          return data[index];
96      }
```

```
97
98      //根据下标修改元素
99      public void set(int index, E e)throws IndexOutOfBoundsException {
100         if (index < 0){
101             throw new IndexOutOfBoundsException("输入的下标小于下界（下界为0）,你
                    传入的下标是" + index + "。");
102         }
103         if (index >= size){
104             String str = "输入的下标超过上界（上界为" + (size - 1) + "）,你传入的
                    下标是" + index + "。";
105             throw new IndexOutOfBoundsException(str);
106         }
107         data[index] = e;
108     }
109
110     //判断元素是否存在
111     public boolean contains(E e){
112         int i = 0;
113         for (; i < size; i++) {
114             if (data[i].equals(e)){
115                 break;
116             }
117         }
118         if (i >= size){
119             return false;
120         }
121         return true;
122     }
123
124     //查找元素位置
125     public int indexOf(E e){
126
127         for (int i = 0; i < size; i++) {
128             if (data[i].equals(e)){
129                 return i;
130             }
131         }
132
133         return -1;
134
135     }
136
137     //从后往前查找元素的位置
138     public int lastIndexOf(E e){
139         for (int i = size - 1; i >=0 ; i--) {
140             if (data[i].equals(e)){
141                 return i;
142             }
143         }
144         return -1;
145     }
```

```
146
147     //清空顺序表
148     public void clear(){
149         size = 0;
150     }
151
152     //表中元素的个数
153     public int size(){
154         return size;
155     }
156
157     //空表判断
158     public boolean isEmpty(){
159         return size == 0;
160     }
161
162     @Override
163     public String toString() {
164         if (size == 0){
165             return "[]";
166         }
167         String str = "[";
168         for (int i = 0; i < size - 1; i++) {
169             str += data[i];
170             str += ",";
171         }
172         str += data[size - 1];
173         str += "]";
174         return str;
175     }
176 }
```

第二章　链表

单向、不带傀儡节点、带环的链表面试常见。

双向、带傀儡节点、带环的链表实际开发中常见。

```java
public class Node<E> {
    public E val;
    public Node<E> next;

    public Node(E val){
        this.val = val;
    }
}
```

2.1　链表的使用

```java
import java.util.Scanner;

public class Test {
    public static Node createList(){
        Node<Integer> a = new Node<>(1);
        Node<Integer> b = new Node<>(2);
        Node<Integer> c = new Node<>(3);
        Node<Integer> d = new Node<>(4);
        a.next = b;
        b.next = c;
        c.next = d;
        d.next = null;
        return a;
    }
    public static void main(String[] args) {
        Node head = createList();

        //遍历链表，打印链表的每个元素
        System.out.println("遍历链表，打印链表的每个元素");
        for (Node tmp = head; tmp != null; tmp = tmp.next) {
            System.out.println(tmp.val);
        }
        System.out.println("----------------------------------------");
        //找链表的最后一个节点
        System.out.println("找链表的最后一个节点");
        Node cur = head;
        while(cur != null && cur.next != null){
            cur = cur.next;
        }
        System.out.println(cur.val);
        System.out.println("----------------------------------------");
        //遍历链表，找到链表的倒数第二个节点
        System.out.println("遍历链表，找到链表的倒数第二个节点");
```

```
34          cur = head;
35          while(cur != null && cur.next != null && cur.next.next != null){
36              cur = cur.next;
37          }
38          System.out.println(cur.val);
39          System.out.println("----------------------------------------");
40          //取链表的第n个节点
41          System.out.println("取链表的第n个节点:");
42          cur = head;
43          int index = 3;
44          for (int i = 1; i < index; i++) {
45              try{
46                  cur = cur.next;
47              }catch(NullPointerException e){
48                  System.out.println("节点超出上限, 该链表共有" + i + "个节点! ");
49                  e.printStackTrace();
50              }
51          }
52          try{
53              System.out.println(cur.val);
54          }catch(NullPointerException e){
55              System.out.println("节点超出上限! ");
56              e.printStackTrace();
57          }
58          System.out.println("----------------------------------------");
59          //获取链表的长度
60          System.out.println("获取链表的长度:");
61          cur = head;
62          int count = 0;
63          for (;cur != null;cur = cur.next) {
64              count++;
65          }
66          System.out.println("该链表共有" + count + "个节点! ");
67          System.out.println("----------------------------------------");
68          //遍历链表, 是否存在某个元素
69          System.out.println("遍历链表, 是否存在某个元素:");
70          cur = head;
71          Scanner scanner = new Scanner(System.in);
72          System.out.println("请输入要查找的元素: ");
73          int findNum = scanner.nextInt();
74          for (;cur != null; cur = cur.next) {
75              if (cur.val.equals(findNum)){
76                  break;
77              }
78          }
79          if (cur != null){
80              System.out.println("找到了! ");
81          }else {
82              System.out.println("没找到! ");
83          }
84      }
```

```
}
```

2.2 链表的插入删除

链表类

```
public class Node {
    int val;
    Node next;
    public Node(int val){
        this.val = val;
    }
}
```

链表的插入删除操作

```
public class Test {
    public static Node creatList(){
        Node a = new Node(1);
        Node b = new Node(2);
        Node c = new Node(3);
        Node d = new Node(4);
        a.next = b;
        b.next = c;
        c.next = d;
        d.next = null;
        return a;
    }

    //按值删除
    public static Node removeValue(Node head, int toDelete){
        if (head == null){
            return head;
        }
        if (head.val == toDelete){
            return head.next;
        }

        Node prev = head;
        while (prev != null && prev.next != null && prev.next.val != toDelete){
            prev = prev.next;
        }
        if (!(prev == null || prev.next == null)) {
            Node deleteNode = prev.next;
            prev.next = deleteNode.next;
            deleteNode.next = null;//回收删除节点的下一个指针
            return head;
        }
        return head;
```

```
34      }
35      //按节点位置删除1
36      public static Node removeNode(Node head,Node toDelete){
37          if (head == null){
38              return head;
39          }
40          if (head == toDelete){
41              return head.next;
42          }
43          Node prev = head;
44          while(prev != null &&  prev.next != toDelete){
45              prev = prev.next;
46          }
47          if (!(prev == null)){
48              prev.next = toDelete.next;
49              toDelete.next = null;
50              toDelete = null;
51              return head;
52          }
53          return head;
54      }
55      //按节点位置删除2
56      public static Node removeNode1(Node head,Node toDelete) {
57          if (head == null){
58              return head;
59          }
60          if (head == toDelete){
61              return head.next;
62          }
63          //无法删除最后一个节点
64          if (toDelete.next != null){
65              toDelete.val = toDelete.next.val;
66              toDelete.next = toDelete.next.next;
67              return head;
68          }
69          return head;
70      }
71      //统计有效值节点的个数
72      public static int size(Node head){
73          int count = 0;
74          for (Node cur = head; cur != null ; cur = cur.next) {
75              count++;
76          }
77          return count;
78      }
79      //按下标删除
80      public static Node removeIndex(Node head, int index){
81          //找到待删除节点的前一个位置index - 1
82          if (index < 0){
83              System.out.println("下标越界, 下界为0。您输入的下界是" + index + "。")
                    ;
```

```
84              return head;
85          }
86          if(index >= size(head)){
87              System.out.println("下标越界，上界为" + size(head) + "。您输入的下界是
                    " + index + "。");
88              return head;
89          }
90          if (index == 0){
91              return head.next;
92          }
93          Node prev = head;
94          for (int i = 0; i < index - 1; i++) {
95              prev = prev.next;
96          }
97          prev.next = prev.next.next;
98          return head;
99      }
100
101     public static Node removeTail(Node head){
102         if (head == null){
103             return head;
104         }
105         Node pre = head;
106         while( pre != null &&pre.next.next!= null && pre.next != null){
107             pre = pre.next;
108         }
109         if (!(pre.next == null)){
110             pre = pre.next;
111             return head;
112         }
113         return head;
114     }
115     //把数组转为链表
116     public static Node arrayToLinkedList(int[] array){
117         Node head = new Node(0);
118         Node tmp = head;
119         for (int i = 0; i < array.length; i++) {
120             Node cur = new Node(array[i]);
121             tmp.next = cur;
122             tmp = tmp.next;
123         }
124         return head.next;
125     }
126
127     public static void main(String[] args) {
128         Scanner scanner = new Scanner(System.in);
129         Node head = creatList();
130         Node insert = new Node(-3);
131         Node pre = null;
132         Node next = null;
133 //        System.out.println("请输入要插入元素的位置（比如在2和3之间插入，则输入2
```

```
    ) ");
//          int index = scanner.nextInt();
//          for(pre = head;pre != null;pre = pre.next){
//              if (index == pre.val){
//                  next = pre.next;
//                  break;
//              }
//          }
//          if (pre != null){
//              insert.next = next;
//              pre.next = insert;
//              insert = null;
//          }else {
//              System.out.println("下标越界, 无法插入! ");
//          }
//          for (pre = head;pre != null;pre = pre.next){
//              System.out.println(pre.val);
//          }
          //节点插入到链表的头部
          insert.next = head;
          head = insert;
          for (pre = head;pre != null;pre = pre.next){
              System.out.println(pre.val);
          }
    }
}
```