Java之美[从菜鸟到高手演变]之数据结构基础、线性表、栈和队列、数组和字符串

一、数据结构概念

用我的理解，数据结构包含数据和结构，通俗一点就是将数据按照一定的结构组合起来，不同的组合方式会有不同的效率，使用不同的场景，如此而已。比如我们最常用的数组，就是一种数据结构，有独特的承载数据的方式，按顺序排列，其特点就是你可以根据下标快速查找元素，但是因为在数组中插入和删除元素会有其它元素较大幅度的便宜，所以会带来较多的消耗，所以因为这种特点，使得**数组适合：查询比较频繁，增、删比较少的情况，这就是数据结构的概念。**数据结构包括**两大类：线性结构和非线性结构，线性结构包括：数组、链表、队列、栈等，非线性结构包括树、图、表等及衍生类结构。**本章我们先讲解线性结构，主要从数组、链表、队列、栈方面进行讨论，非线性数据结构在后面会继续讲解。

二、线性表

线性表是最基本、最简单、也是最常用的一种数据结构。线性表中数据元素之间的关系是一对一的关系，即除了第一个和最后一个数据元素之外，其它数据元素都是首尾相接的。线性表的逻辑结构简单，便于实现和操作。因此，线性表这种数据结构在实际应用中是广泛采用的一种数据结构。其基本操作主要有：

1）MakeEmpty(L) 这是一个将L变为空表的方法 2）Length（L） 返回表L的长度，即表中元素个数 3）Get（L，i） 这是一个函数，函数值为L中位置i处的元素（1≤i≤n） 4）Prev（L，i） 取i的前驱元素 5）Next（L，i） 取i的后继元素 6）Locate（L，x） 这是一个函数，函数值为元素x在L中的位置 7）Insert（L，i，x）在表L的位置i处插入元素x，将原占据位置i的元素及后面的元素都向后推一个位置 8）Delete（L，p） 从表L中删除位置p处的元素 9）IsEmpty(L) 如果表L为空表(长度为0)则返回true，否则返回false 10）Clear（L）清除所有元素 11）Init（L）同第一个，初始化线性表为空 12）Traverse（L）遍历输出所有元素 13）Find（L，x）查找并返回元素 14）Update（L，x）修改元素 15）Sort（L）对所有元素重新按给定的条件排序 16) strstr(string1,string2)用于字符数组的求string1中出现string2的首地址

不管采用哪种方式实现线性表，至少都应该具有上述这些基本方法，下面我会将下数据结构的基本实现方式。

三、基础数据结构

数据结构是一种抽象的数据类型（ADT），可以这么说，我们可以采用任意的方式实现某种数据结构，只要符合将要实现的数据结构的特点，数据结构就是一种标准，我们可以采用不同的方式去实现，最常用的两种就是数组和链表（包括单链表、双向链表等）。数组是非常常见的数据类型，在任何一种语言里都有它的实现，我们这里采用Java来简单实现一下数组。 数组是一种引用类型的对象，我们可以像下面这样的方式来声明数组：

```
int a[];
int[] b;
int []c;
```

```
a = new int[10];
```

总结起来，声明一个数组有基本的三个因素：类型、名称、下标，Java里，数组在格式上相对灵活，下标和名称可以互换位置，前三种情况我们可以理解为声明一个变量，后一种为其赋值。或者像下面这样，在声明的时候赋值：

```
int c[] = {2,3,6,10,99};
int []d = new int[10];
```

我稍微解释一下，其实如果只执行：int[] b,只是在栈上创建一个引用变量，并未赋值，只有当执行d = new int[10]才会在堆上真正的分配空间。上述第一行为静态初始化，就是说用户指定数组的内容，有系统计算数组的大小，第二行恰恰相反，用户指定数组的大小，由系统分配初始值，我们打印一下数组的初始值：

```
int []d = new int[10];
System.out.println(d[2]);
```

结果输出0，对于int类型的数组，默认的初始值为0.

但是，绝对不可以像下面这样：

```
int e[10] = new int[10];
```

无法通过编译，至于为什么，语法就是这样，这是一种规范，不用去想它。????? 我们可以通过下标来检索数组。下面我举个简单的例子，来说明下数组的用法。

```
public static void main(String[] args) {

    String name[];

    name = new String[5];
    name[0] = "egg";
    name[1] = "erqing";
    name[2] = "baby";

    for (int i = 0; i < name.length; i++) {
        System.out.println(name[i]);
    }
}
```

这是最简单的数组声明、创建、赋值、遍历的例子，下面写个增删的例子。

```
package com.xtfggef.algo.array;

public class Array {

    public static void main(String[] args) {
        int value[] = new int[10];
        for (int i = 0; i < 10; i++) {
            value[i] = i;
        }

        // traverse(value);
        // insert(value, 666, 5);
        delete(value, 3);
        traverse(value);

    }
```

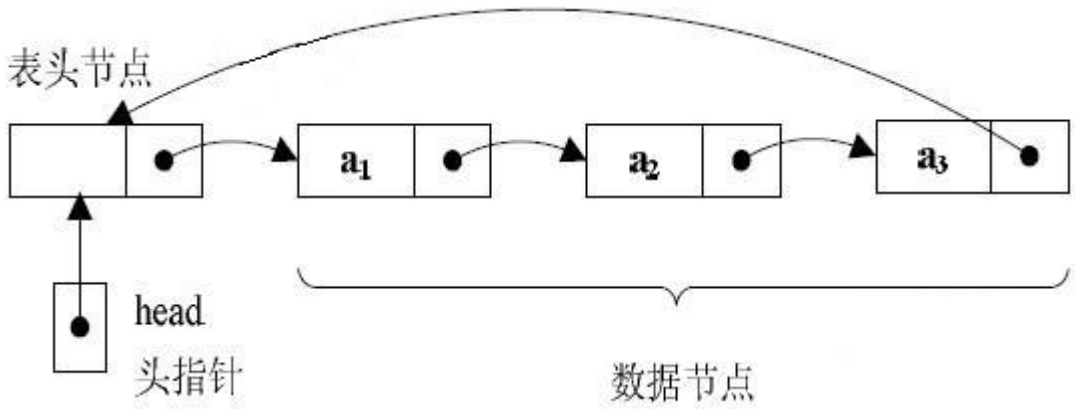
```
    public static int[] insert(int[] old, int value, int index) {
        for (int k = old.length - 1; k > index; k--)
            old[k] = old[k - 1];
        old[index] = value;
        return old;
    }

    public static void traverse(int data[]) {
        for (int j = 0; j < data.length; j++)
            System.out.print(data[j] + " ");
    }

    public static int[] delete(int[] old, int index) {
        for (int h = index; h < old.length - 1; h++) {
            old[h] = old[h + 1];
        }
        old[old.length - 1] = 0;
        return old;
    }
}
```

简单写一下，主要想说明数组中删除和增加元素的原理：增加元素，需要将index后面的依次向后移动，然后将值插入index位置，删除则将后面的值依次向前移动，较简单。

要记住：数组是表示相同类型的一类数据的集合，下标从0开始，就行了。

数组实现的线下表可以参考ArrayList，在JDK中附有源码，感兴趣的同学可以读读。下面我简单介绍下单链表。

单链表是最简单的链表，有节点之间首尾连接而成，简单示意如下：



除了头节点，每个节点包含一个数据域一个指针域，除了头、尾节点，每个节点的指针指向下一个节点，下面我们写个例子操作一下单链表。

```
package com.xtfggef.algo.linkedlist;

public class LinkedList<T> {

    /**
     * class node
     * @author egg
     * @param <T>
```

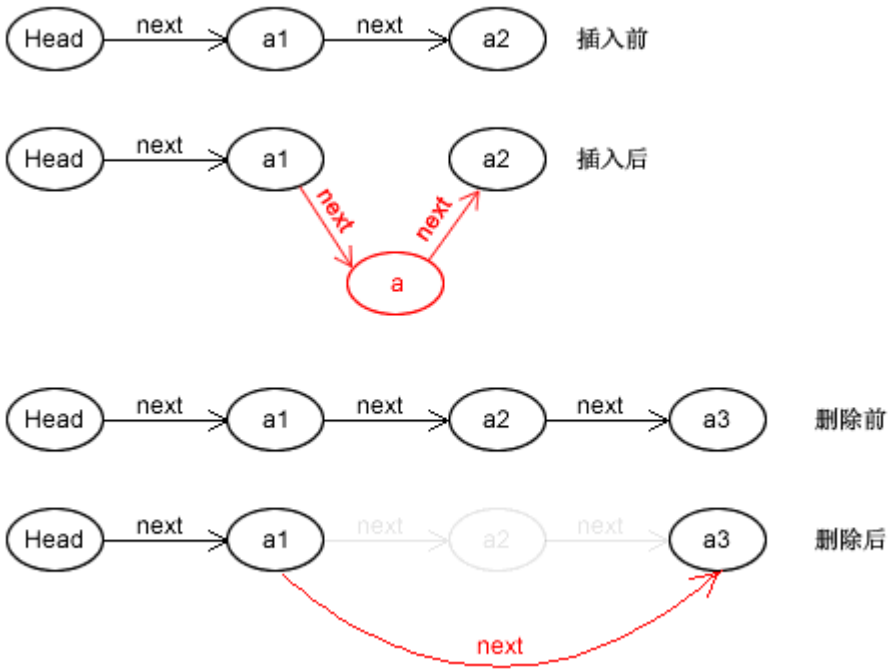
```
     */
    private static class Node<T> {
        T data;
        Node<T> next;

        Node(T data, Node<T> next) {
            this.data = data;
            this.next = next;
        }

        Node(T data) {
            this(data, null);
        }
    }

    // data
    private Node<T> head, tail;

    public LinkedList() {
        head = tail = null;
    }

    /**
     * judge the list is empty
     */
    public boolean isEmpty() {
        return head == null;
    }

    /**
     * add head node
     */
    public void addHead(T item) {
        head = new Node<T>(item);
        if (tail == null)
            tail = head;
    }

    /**
     * add the tail pointer
     */
    public void addTail(T item) {
        if (!isEmpty()) {
            tail.next = new Node<T>(item);
            tail = tail.next;
        } else {
            head = tail = new Node<T>(item);
        }
    }

    /**
     * print the list

     */
```

```java
    public void traverse() {
        if (isEmpty()) {
            System.out.println("null");
        } else {
            for (Node<T> p = head; p != null; p = p.next)
                System.out.println(p.data);
        }
    }

    /**
     * insert node from head
     */
    public void addFromHead(T item) {
        Node<T> newNode = new Node<T>(item);
        newNode.next = head;
        head = newNode;
    }

    /**
     * insert node from tail
     */
    public void addFromTail(T item) {
        Node<T> newNode = new Node<T>(item);
        Node<T> p = head;
        while (p.next != null)
            p = p.next;
        p.next = newNode;
        newNode.next = null;
    }

    /**
     * delete node from head
     */
    public void removeFromHead() {
        if (!isEmpty())
            head = head.next;
        else
            System.out.println("The list have been emptied!");
    }

    /**
     * delete frem tail, lower effect
     */
    public void removeFromTail() {
        Node<T> prev = null, curr = head;
        while (curr.next != null) {
            prev = curr;
            curr = curr.next;
            if (curr.next == null)
                prev.next = null;
        }
    }
```

```java
    /**
     * insert a new node
     * @param appointedItem
     * @param item
     * @return
     */
    public boolean insert(T appointedItem, T item) {
        Node<T> prev = head, curr = head.next, newNode;
        newNode = new Node<T>(item);
        if (!isEmpty()) {
            while ((curr != null) && (!appointedItem.equals(curr.data))) {
                prev = curr;
                curr = curr.next;
            }
            newNode.next = curr;
            prev.next = newNode;
            return true;
        }
        return false;
    }

    public void remove(T item) {
        Node<T> curr = head, prev = null;
        boolean found = false;
        while (curr != null && !found) {
            if (item.equals(curr.data)) {
                if (prev == null)
                    removeFromHead();
                else
                    prev.next = curr.next;
                found = true;
            } else {
                prev = curr;
                curr = curr.next;
            }
        }
    }


    public int indexOf(T item) {
        int index = 0;
        Node<T> p;
        for (p = head; p != null; p = p.next) {
            if (item.equals(p.data))
                return index;
            index++;

        }
        return -1;
    }

    /**

     * judge the list contains one data
```

```
     */
    public boolean contains(T item) {
        return indexOf(item) != -1;
    }
}
```

单链表最好玩儿的也就是增加和删除节点，下面的两个图分别是用图来表示单链表增、删节点示意，看着图学习，理解起来更加容易！



接下来的队列和栈，我们分别用不同的结构来实现，队列用数组，栈用单列表，读者朋友对此感兴趣，可以分别再用不同的方法实现。

四、队列

队列是一个常用的数据结构，是一种先进先出（First In First Out, FIFO）的结构，也就是说只能在表头进行删除，在表尾进行添加，下面我们实现一个简单的队列。

```
package com.xtfggef.algo.queue;

import java.util.Arrays;

public class Queue<T> {

    private int DEFAULT_SIZE = 10;

    private int capacity;

    private Object[] elementData;

    private int front = 0;
    private int rear = 0;
```

```
    public Queue()
    {
        capacity = DEFAULT_SIZE;
        elementData = new Object[capacity];
    }

    public Queue(T element)
    {
        this();
        elementData[0] = element;
        rear++;
    }

    public Queue(T element , int initSize)
    {
        this.capacity = initSize;
        elementData = new Object[capacity];
        elementData[0] = element;
        rear++;
    }

    public int size()
    {
        return rear - front;
    }

    public void add(T element)
    {
        if (rear > capacity - 1)
        {
            throw new IndexOutOfBoundsException("the queue is full!");
        }
        elementData[rear++] = element;
    }

    public T remove()
    {
        if (empty())
        {
            throw new IndexOutOfBoundsException("queue is empty");
        }

        @SuppressWarnings("unchecked")
        T oldValue = (T)elementData[front];

        elementData[front++] = null;
        return oldValue;
    }

    @SuppressWarnings("unchecked")
    public T element()
    {

        if (empty())
```

```
        {
            throw new IndexOutOfBoundsException("queue is empty");
        }
        return (T)elementData[front];
    }

    public boolean empty()
    {
        return rear == front;
    }

    public void clear()
    {

        Arrays.fill(elementData , null);
        front = 0;
        rear = 0;
    }
    public String toString()
    {
        if (empty())
        {
            return "[]";
        }
        else
        {
            StringBuilder sb = new StringBuilder("[");
            for (int i = front  ; i < rear ; i++ )
            {
                sb.append(elementData[i].toString() + ", ");
            }
            int len = sb.length();
            return sb.delete(len - 2 , len).append("]").toString();
        }
    }
    public static void main(String[] args){
        Queue<String> queue = new Queue<String>("ABC", 20);
        queue.add("DEF");
        queue.add("egg");
        System.out.println(queue.empty());
        System.out.println(queue.size());
        System.out.println(queue.element());
        queue.clear();
        System.out.println(queue.empty());
        System.out.println(queue.size());
    }
}
```

队列只能在表头进行删除，在表尾进行增加，这种结构的特点，适用于排队系统。

五、栈

栈是一种后进先出（Last In First Out，LIFO）的数据结构，我们采用单链表实现一个栈。

```java
package com.xtfggef.algo.stack;

import com.xtfggef.algo.linkedlist.LinkedList;

public class Stack<T> {

    static class Node<T> {
        T data;
        Node<T> next;

        Node(T data, Node<T> next) {
            this.data = data;
            this.next = next;
        }

        Node(T data) {
            this(data, null);
        }
    }

    @SuppressWarnings("rawtypes")
    static LinkedList list = new LinkedList();

    @SuppressWarnings("unchecked")
    public T push(T item) {
        list.addFromHead(item);
        return item;
    }

    public void pop() {
        list.removeFromHead();
    }

    public boolean empty() {
        return list.isEmpty();
    }

    public int search(T t) {
        return list.indexOf(t);
    }

    public static void main(String[] args) {
        Stack<String> stack = new Stack<String>();
        System.out.println(stack.empty());
        stack.push("abc");
        stack.push("def");
        stack.push("egg");
        stack.pop();
        System.out.println(stack.search("def"));
    }
}
```