# 作业四 线性表的链式表示1

## 1、什么是线性表的链式存储？线性表的链式存储如何实现？

线性表的链式存储是一种数据结构，它使用任意的存储单元依次存放线性表的节点。这些存储单元可以是连续的，也可以是不连续的，甚至是零散的分布在内存中的任意位置上。因此，链表中的节点的逻辑次序和物理次序不一定相同。为了正确表示节点间的逻辑结构，在存储每个节点值的同时，还必须存储指示其后继节点的地址（或位置）信息。这两部分组成了链表中的节点结构。

在 Java 中实现线性表的链式存储，可以通过创建一个内部类 `Node` 来代表链表的节点。每个 `Node` 对象都有一个数据域和一个指针域。数据域用来存放节点的值，指针域用来存放后继节点的地址。链表正是通过每个节点的链域将线性表的 n 个节点按其逻辑次序连接在一起的。

在 Java 的标准库中，`java.util.LinkedList` 类提供了双向链表的实现。[LinkedList (Java Platform SE 8 ) (oracle.com)](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html)

## 2、线性表的链式存储中各结点由哪些部分组成？

线性表的链式存储中的每个节点通常由两部分组成：

* **数据域：**用于存储节点的值。
* **指针域（也称为链域）：**用于存储后继节点的地址。

这两部分组成了链表中的节点结构。数据域存放节点的数据信息，而指针域则用于链接下一个节点，从而形成一个链式的结构。这种结构使得链表在插入和删除操作时，只需修改相邻节点的指针，效率较高。但是，链表的访问需要从头节点开始，按链域逐个访问，所以访问效率相对较低。这是链表的主要特点和优缺点。

## 3、线性表的链式存储有哪几种方式？

1. **单链表：**每个节点只有一个指向直接后继节点的指针域²。单链表中，任何两个元素的存储位置之间没有固定的联系，每个元素的存储位置都包含在其直接前驱节点的信息中¹。因此，单链表是非随机存取的存储结构¹。
2. **单循环链表：**单循环链表是一种特殊的单链表，它的最后一个节点的指针域不再为空，而是指向链表的头节点，形成一个环形的链表²。
3. **双向循环链表：**双向循环链表是一种更复杂的链式存储结构，每个节点有两个指针，一个指向直接后继节点，另一个指向直接前驱节点²。双向循环链表的最后一个节点的后继指针指向头节点，头节点的前驱指针指向最后一个节点，形成一个环形的链表²。

## 4、什么是头引用、头结点和首元结点？

头引用、头结点和首元结点是链表中的三个重要概念：

1. **头引用（头指针）：**它是指向链表中第一个节点的指针。如果链表有头结点，则头引用指向头结点；如果链表没有头结点，则头引用指向首元结点。

2. **头结点：**它是在首元结点之前附设的一个节点，其指针域指向首元结点。头结点的数据域可以不存储任何信息，也可以存储与数据元素类型的其他附加信息，例如，当数据元素为整数型时，头结点的数据域中可存放该线性表的长度。

3. **首元结点：**就是指链表中存储第一个数据元素的结点。如果链表有头结点，则首元结点为头结点的下一个结点；如果链表没有头结点，则首元结点就是链表的第一个结点。

 这三个概念的区别主要在于它们的性质、目的和存在情况。头结点对于单链表来说，可有可无，但为了操作方便，一般情况下单链表都具有头结点。头结点的存在使得对链表的第一个数据元素的操作与其他数据元素相同，无需进行特殊处理，便于空表的和非空表的统一处理。

## 5、请简述链表的特点。

1. **动态大小：**链表的大小可以在运行时动态改变，无需预先分配空间。

2. **插入和删除效率高：**链表支持高效的插入和删除操作，不需要移动其他元素。

3. **不连续内存：**链表的节点可以在内存中分布不连续，不受固定内存大小的限制。

4. **查找效率较低：**链表的查找需要从头遍历，与数组类似，越长速度越慢。

5. **空间利用率高：**链表的空间利用率相对较高，因为它只需要为实际数据分配空间。

6. **灵活性高：**链表可以方便地进行各种复杂的插入和删除操作。

 这些特点使得链表在许多场合下成为了优选的数据结构，尤其是在需要频繁插入和删除元素的情况下。但是，链表的查找效率较低，所以在需要频繁查找的应用中，可能不如数组或其他数据结构效率高。

## 6、请解释以下代码的含义：

public class StuNode{

 private Student student;

 private StuNode nextlink;

 public Student getStudent(){

 return student;

 }

 public void setStudent(Student student) {

 this.student = student;}

 public StuNode getNext() {

 return nextlink;}

 public void setNext(StuNode nextlink) {

 this.nextlink = nextlink;}

}

 public class StuLinkList {

 private StuNode head;

 public StuLinkList(StuNode hd){

 head = hd;

 }

 }

这段代码定义了两个类：`StuNode` 和 `StuLinkList`。

* `StuNode` 类代表一个学生节点，每个节点包含一个 `Student` 对象和一个指向下一个 `StuNode` 的链接。这个类有四个方法：
	+ `getStudent()`：返回当前节点的 `Student` 对象。
	+ `setStudent(Student student)`：设置当前节点的 `Student` 对象。
	+ `getNext()`：返回指向下一个 `StuNode` 的链接。
	+ `setNext(StuNode nextlink)`：设置指向下一个 `StuNode` 的链接。
* `StuLinkList` 类代表一个学生链表，它包含一个 `StuNode` 对象，作为链表的头节点。这个类有一个构造函数，用于初始化链表的头节点。

 总的来说，这段代码是用来实现一个单链表数据结构的，其中每个节点包含一个 `Student` 对象。这种数据结构常用于实现各种算法和数据操作，例如排序、搜索等。

## 7、请简述单链表是如何实现在p引用所指向的结点前后分别插入一个数据元素x的。

在单链表中，可以通过以下步骤在p引用所指向的结点前后分别插入一个数据元素x：

1. 在p之前插入元素x：
	* 创建一个新的结点newNode，并将数据元素x存储在newNode中。
	* 将newNode的next指向p所指向的结点。
	* 将p的前一个结点的next指向newNode。
2. 在p之后插入元素x：
	* 创建一个新的结点newNode，并将数据元素x存储在newNode中。
	* 将newNode的next指向p所指向的结点的next。
	* 将p所指向的结点的next指向newNode。

以下是使用Java实现的示例代码：

public class ListNode {

 int val;

 ListNode next;

 ListNode(int x) { val = x; }

}

public class LinkedList {

 ListNode head;

 public void insertBefore(ListNode p, int x) {

 ListNode newNode = new ListNode(x);

 if (head == p) {

 newNode.next = head;

 head = newNode;

 } else {

 ListNode current = head;

 while (current != null && current.next != p) {

 current = current.next;

 }

 if (current != null) {

 newNode.next = p;

 current.next = newNode;

 }

 }

 }

 public void insertAfter(ListNode p, int x) {

 ListNode newNode = new ListNode(x);

 if (p != null) {

 newNode.next = p.next;

 p.next = newNode;

 }

 }

}

这个代码假设已经有了对应的p结点的引用。如果没有这个引用，可能需要遍历链表来找到这个结点。此外，这个代码没有处理错误情况，例如p不在链表中的情况。在实际使用时，可能需要添加适当的错误检查和处理。

## 8、请解释以下代码的含义：

 public void insertAfter(StuNode p,StuNode s){

 s.setNext(p.getNext());

 p.setNext(s);

 }

这段代码是在一个链表中插入节点的操作。具体来说，它是在节点 p 后面插入一个新的节点 s，这形成了一个单向循环链表。下面是每行代码的详细解释：

1. s.setNext(p.getNext());：这行代码将节点 p 后面的节点设置为节点 s 的下一个节点。也就是说，节点 s 的 next 指针指向了原来节点 p 的 next 指针所指向的节点。
2. p.setNext(s);：这行代码将节点 p 的 next 指针指向节点 s。也就是说，节点 s 被插入到了节点 p 后面。

总的来说，这段代码的作用就是将节点 s 插入到链表中节点 p 的后面。在插入节点 s 之后，节点 p 的下一个节点是 s，而节点 s 的下一个节点是原来节点 p 的下一个节点。这样就完成了在链表中插入节点的操作。

## 9、请解释以下代码的含义：

 public void insert(StuNode p,StuNode s){

 if(head==null) return;

 StuNode q = head;

 while(q.getNext()!=null){

 if(q.getNext()==p){

 q.setNext(s);

 s.setNext(p);

 break;}

 q=q.getNext();}

}

这段代码是一个插入操作的函数，用于在链表中插入一个新的节点。以下是这段代码的详细解释：

public void insert(StuNode p,StuNode s): 这是一个公共方法，名为insert，它接收两个参数，p和s，它们都是StuNode类型的对象。p是我们要在其之前插入新节点的节点，s是我们要插入的新节点。

if(head==null) return;: 这行代码检查链表是否为空（即头节点head是否为null）。如果链表为空，则函数返回，不执行任何操作。

StuNode q = head;: 这行代码创建了一个新的StuNode对象q，并将其初始化为头节点head。q将用于遍历链表。

while(q.getNext()!=null): 这是一个循环，用于遍历链表。只要节点q的下一个节点不为null，循环就会继续。

if(q.getNext()==p): 这是一个条件语句，检查节点q的下一个节点是否是节点p。如果是，那么我们找到了要在其之前插入新节点的位置。

q.setNext(s);: 这行代码将节点q的下一个节点设置为新节点s，这样新节点s就被插入到了节点q和节点p之间。

s.setNext(p);: 这行代码将新节点s的下一个节点设置为节点p，确保链表的连续性。

break;: 这行代码跳出循环，因为我们已经找到了插入新节点的位置，并完成了插入操作。

q=q.getNext();: 这行代码将节点q移动到下一个节点，以便继续遍历链表。

总的来说，这个insert函数的作用是在链表中找到特定的节点p，并在其之前插入一个新的节点s。如果链表为空或者没有找到节点p，则不会插入新节点。这是链表数据结构中常见的一种操作。