

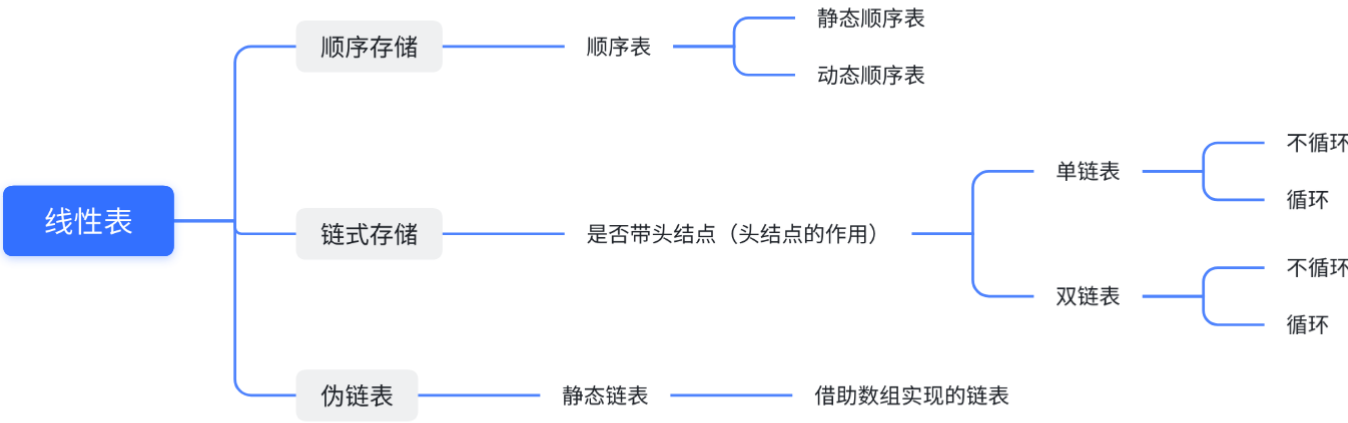
# 2-线性表

C生万物 ● 大道至简 ● 鲍鱼科技+v(15339278619)


## 1、目标

- 1、线性表的概念
- 2、线性表分类
- 3、线性表的实现（难点），至少能够理论画图
- 4、顺序存储和链式存储的特性对比（重点）
- 5、能看懂静态链表的表示形式

线性表由于代码量少，逻辑清晰，复杂度容易刻画，因此线性表往往是**出题的重点**，如果具有编程基础，一定要自己实现一遍代码



## 2、基础概念

 **线性结构**：有一组相同数据类型的元素，有一个被称为第一的元素，也有一个被称为最后一个元素，除了第一元素没有前驱元素，最后一个没有后继元素之外，其余的元素**有且仅有**唯一的前驱和后继元素，这样的结构就称为线性结构

### 3、顺序表

 **什么是顺序表：**

顺序表是线性表的顺序实现，说白了，就是使用**数组**实现线性表。只是数组要被结构体进行管理，而并非直接对赤裸裸的数组进行操作。

 **顺序表的结构特点：**

- 1、逻辑位置相邻，物理位置也相邻
- 2、可以随机访问元素，由于是数组，只需要下标就可以直接访问，速度快
- 3、存储效率高，一个空间全部用于存储数据
- 4、最大的缺点就是插入或删除元素，需要移动数据

**结构定义：**

- **静态顺序表：**

```
1 typedef struct SeqList
2 {
3     ElemType data[N];    //空间大小固定，不可以改变
4     int size;            //元素个数
5 }SeqList;
```

- **动态顺序表：**

```
1 typedef struct SeqList
2 {
3     ElemType *data;    //需要动态开辟空间
4     int capacity;      //容量
5     int size;          //元素个数
6 }SeqList;
```

- 顺序表ADT:

```
1 #define SeqListElem_Type int
2 //定义顺序表数据结构
3 typedef struct SeqList
4 {
5     SeqListElem_Type *base; //顺序表空间指针
6     size_t capacity;        //容量
7     size_t size;            //元素个数
8 }SeqList;
9
10 //函数申明
11 void SeqListInit(SeqList *plist);
12 void SeqListPushBack(SeqList *plist, SeqListElem_Type v);
13 void SeqListPushFront(SeqList *plist, SeqListElem_Type v);
14 void SeqListPopBack(SeqList *plist);
15 void SeqListPopFront(SeqList *plist);
16 size_t SeqListLength(SeqList *plist);
17 size_t SeqListCapacity(SeqList *plist);
18 SeqListElem_Type SeqListFront(SeqList *plist);
19 SeqListElem_Type SeqListBack(SeqList *plist);
20
21 void SeqListInsertByPos(SeqList *plist, int pos, SeqListElem_Type v);
22 int SeqListFind(SeqList *plist, SeqListElem_Type key);
23
24 void SeqListDeleteByVal(SeqList *plist, SeqListElem_Type key);
25
26 void SeqListSort(SeqList *plist);
27
28 void SeqListReverse(SeqList *plist);
29 void SeqListMerge(SeqList *plist, SeqList *pa, SeqList *pb);
30
31 void SeqListShow(SeqList *plist);
32
33 void SeqListClear(SeqList *plist);
34 void SeqListDestroy(SeqList *plist);
35
36 //顺序表辅助函数
37 bool SeqListFull(SeqList *plist);
38 bool SeqListEmpty(SeqList *plist);
39 bool _Inc(SeqList *plist);
```

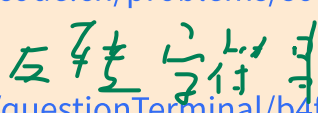


通过实现上述顺序表接口，需要达到的目的：

1、深入掌握顺序表是如何管理的

- 2、顺序表有空间的限制，所以插入数据要判满，删除数据要判空
- 3、插入删除有可能移动数据
- 4、空间不足时，顺序表会自动增长空间，掌握空间增长的实现
- 5、要学会顺序表操作的各种画图，其中理解顺序表的管理是核心关键

#### 顺序表必会题型：

- 1、两个有序顺序表的合并 <https://leetcode.cn/problems/merge-sorted-array/>
- 2、顺序表的排序 <https://leetcode.cn/problems/sort-an-array/>
- 3、顺序表的逆置  
  
<https://www.nowcoder.com/questionTerminal/b4f598480524493aae4686947fbf31dc?>
- 4、顺序表的二分查找 <https://leetcode.cn/problems/binary-search/>
- 5、顺序表的旋转 <https://leetcode.cn/problems/rotate-array/>

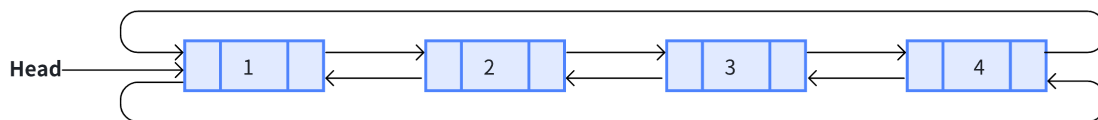
## 4、链表

#### 链表概述：

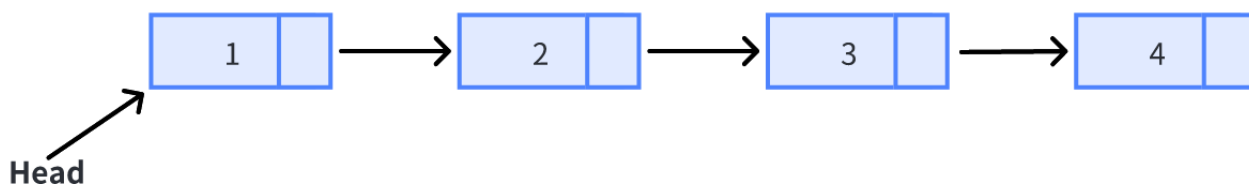
- 1、组成链表的三个关键指标：**是否带头节点、单链表还是双链表、是否循环**  
所以链表的组合一共有8种基本情况，
- 2、其次还需要特别关注链表的**管理方式**：  
有的只是通过一个头指针  
有的除了头指针还有尾指针，  
有的指针是封装在结构体内部，通过结构体变量进行操作
- 3、对链表的操作，需要达到灵活处理，关键点在于对节点指针的理解，只要能够根据代码正确画图，一般链表的问题就可以迎刃而解，也可以检验代码的正确性
- 4、链表的操作难点在于要能够掌握指针，能够区分谁的next是谁，prev是谁，  
甚至next->next, prev->prev, prev->next, next->prev到底代表的是哪一个指针
- 5、链表容易出选择题，让选择合适的插入或删除语句，只要能分清指针，会画图，肯定能做对
- 6、有编程基础的前提，一定要自己实现一遍各种链表的代码

## 头结点作用：

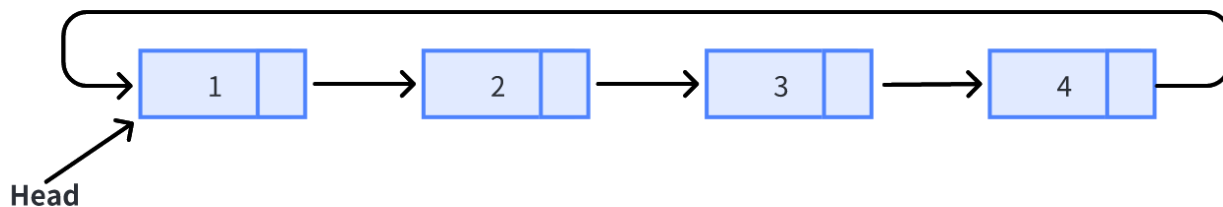
由于不修改头指针的指向，所以头结点最大的作用在于：可以让后续的节点进行统一处理



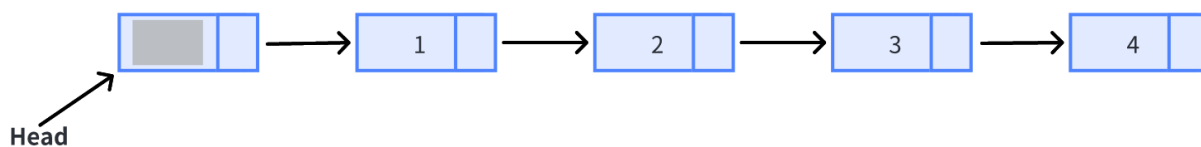
### 1、不带头单链表



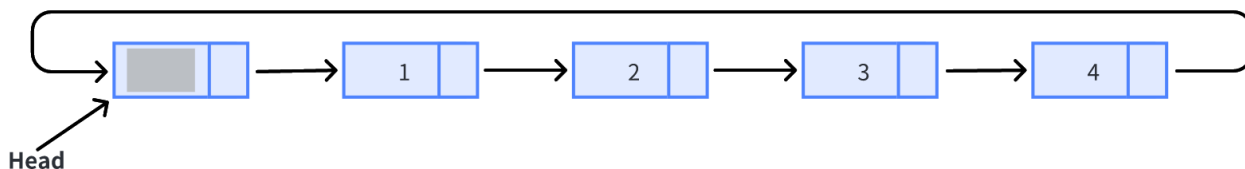
### 2、不带头循环单链表



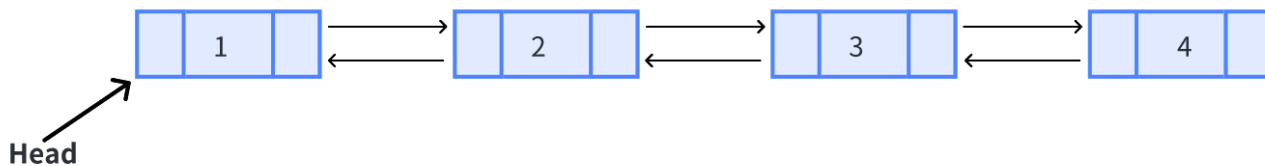
### 3、带头单链表



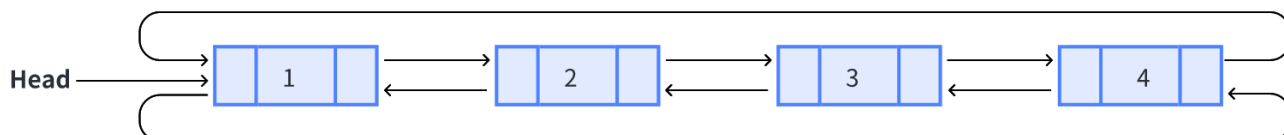
### 4、带头循环单链表



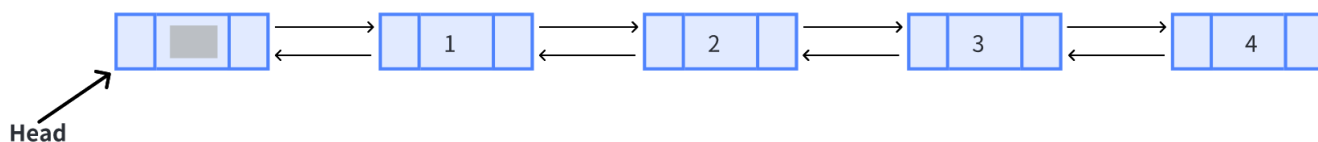
### 5、不带头双链表



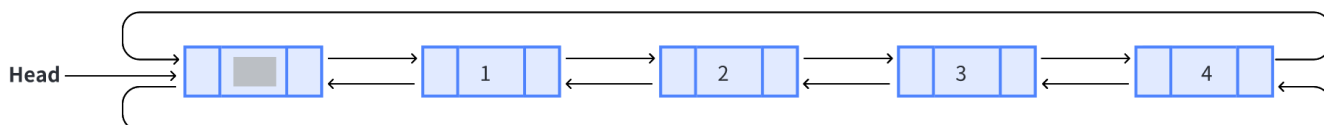
## 6、不带头循环双链表




## 7、带头双链表



## 8、带头循环双链表



 一共有8种链表结构，其中不带头结点的操作最容易出错，因为对第一个节点的插入和删除，会导致需要修改头指针，这往往是最容易被忽略掉的一种情形，一定要注意。

## • 链表ADT

```


1 #define ListElem_Type int
2
3 //定义单链表节点
4 typedef struct ListNode
5 {
6     ListElem_Type data;
7     struct ListNode *next;
8 }ListNode;
9
10 //定义单链表类型
11 typedef ListNode* SList;

```

```

12
13 //函数申明
14 void ListInit(SList *phead);
15 void ListPushBack(SList *phead, SListElem_Type v);
16 void ListPushFront(SList *phead, SListElem_Type v);
17 void ListPopBack(SList *phead);
18 void ListPopFront(SList *phead);
19 void ListShow(SList phead);
20
21 size_t ListLength(SList phead);
22 SListNode* ListFind(SList phead, SListElem_Type key);
23 void ListInsert(SList *phead, ListNode *pos);
24 void ListDeleteByVal(SList *phead, SListElem_Type key);
25
26 void ListClear(SList *phead);
27 void ListDestroy(SList *phead);
28
29 void SListReverse(SList *phead);
30 void SListSort(SList *phead);

```

 通过实现上述链表接口，需要达到的目的：

- 1、深入掌握链表是如何管理的
- 2、链表没有空间的限制，所以插入数据无需判满
- 3、插入删除不需要移动数据
- 4、链表对空间的使用是精准控制，需要多少开辟多少
- 5、要学会链表操作的各种画图，其中理解链表的管理是核心关键

 链表必会题型：

- 1、链表的反转 <https://leetcode.cn/problems/reverse-linked-list/>
- 2、两个有序链表的合并 <https://leetcode.cn/problems/merge-two-sorted-lists/>
- 3、返回倒数第k个节点 <https://leetcode.cn/problems/kth-node-from-end-of-list-lcci/>
- 4、查找两个链表的第一个相交节点 <https://leetcode.cn/problems/intersection-of-two-linked-lists/description/>
- 5、删除指定链表节点 <https://leetcode.cn/problems/remove-linked-list-elements/description/>
- 6、链表的排序 <https://leetcode.cn/problems/insertion-sort-list/>

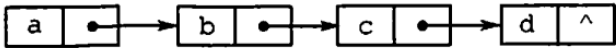
5、顺序表VS链表

	存取方式	逻辑结构	物理结构	插入、删除
顺序表	随机存取	顺序结构	空间连续	需要移动元素
链表	顺序存取	顺序结构	空间不连续	不需要移动，只需修改指针


6、静态链表

0		2
1	b	6
2	a	1
3	d	-1
4		
5		
6	c	3

(a)静态链表示例



(b)静态链表对应的单链表

 静态链表需要能够看懂静态链表的表示形式，是如何通过数组形式表示出链表的结构。这往往也是考试过程中经常出现的题型形式。

25. 【2016 统考真题】已知表头元素为 c 的单链表在内存中的存储状态如下表所示。

地址	元素	链接地址
1000H	a	1010H
1004H	b	100CH
1008H	c	1000H
100CH	d	NULL
1010H	e	1004H
1014H		

现将 f 存放于 1014H 处并插入单链表, 若 f 在逻辑上位于 a 和 e 之间, 则 a, e, f 的“链接地址”依次是 ( )。

- A. 1010H, 1014H, 1004H
- B. 1010H, 1004H, 1014H
- C. 1014H, 1010H, 1004H
- D. 1014H, 1004H, 1010H