

# 链表(Linked List)详解

想不出来起什么名字 已于 2025-01-03 21:09:39 修改

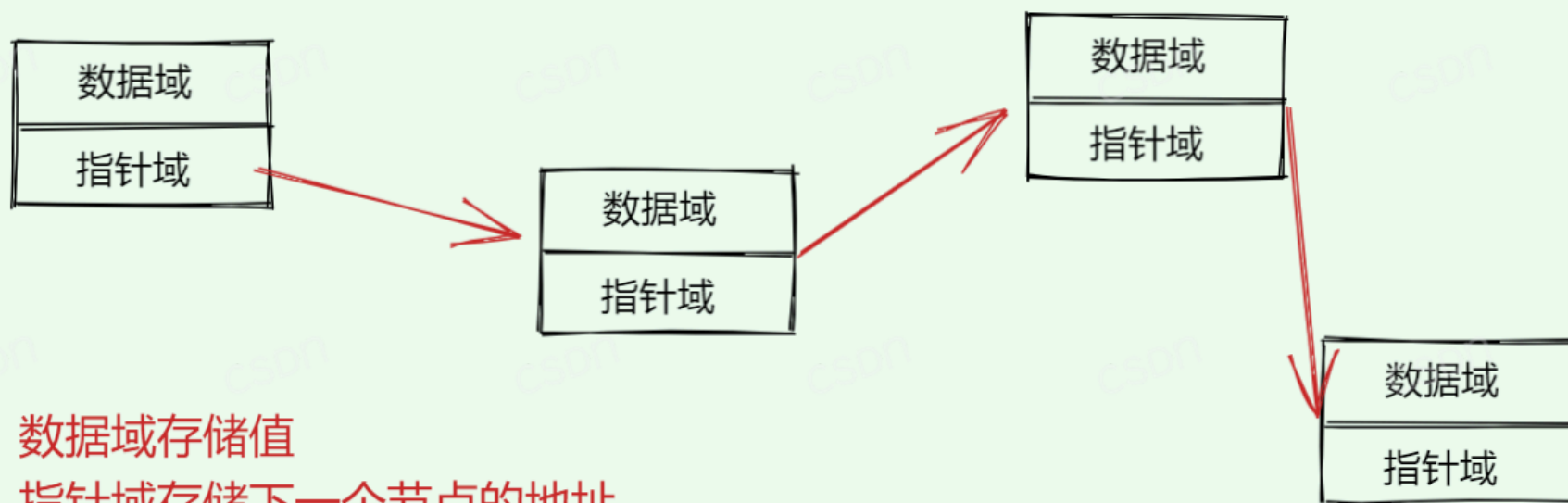
## 引言

上一篇我们提到 **顺序表** 这个 **数据结构**，它的底层是数组

而这一篇，我们学习一个新的数据结构——链表

和顺序表不同，链表在**逻辑上是连续的**，但是在物理上（内存分布）可能是不连续的

我们人为抽象链表的逻辑是连续的



数据域存储值

指针域存储下一个节点的地址

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

我们可以将链表想象成火车，一节一节车厢就是链表的节点，而车厢里的内容就是数据域，链接每个车厢的车钩就相当于指针



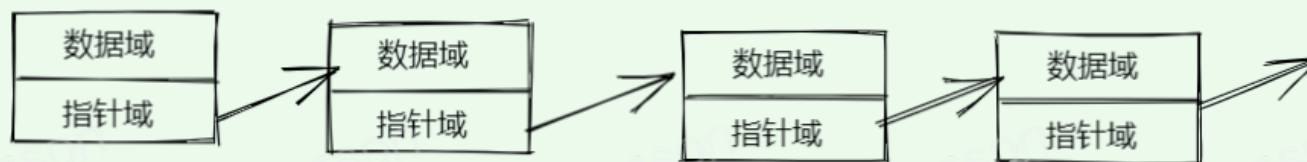
链表的每个节点是我们自己在堆上开辟的，在堆上开辟的空间，操作系统是按照一定的策略开辟的，两次开辟的空间地址可能连续，也可能不连续，所以在物理结构上是不连续的

这里就可以总结出链表最大的特点：**数据做岛屿，指针做桥梁**，这样分散的数据就有了联系

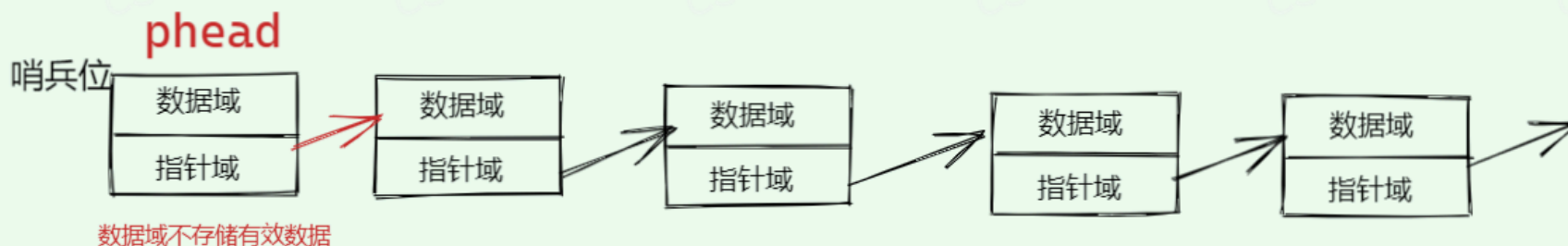
## 链表的分类

基于带头与否，循环与否，双向与否，我们可以组合出 $2^3 = 8$ 种链表

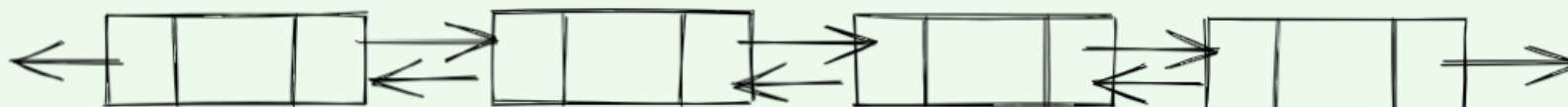
不带头



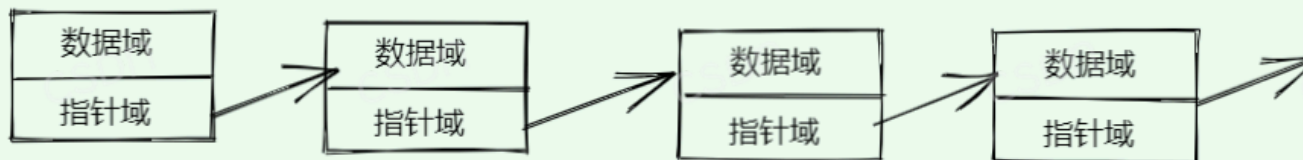
带头



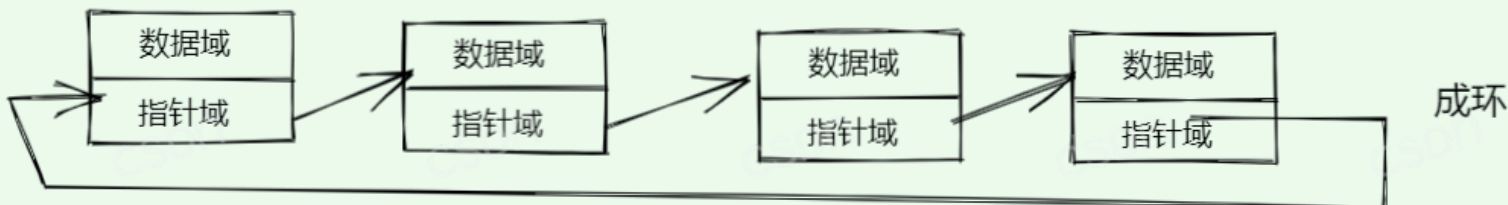
双向



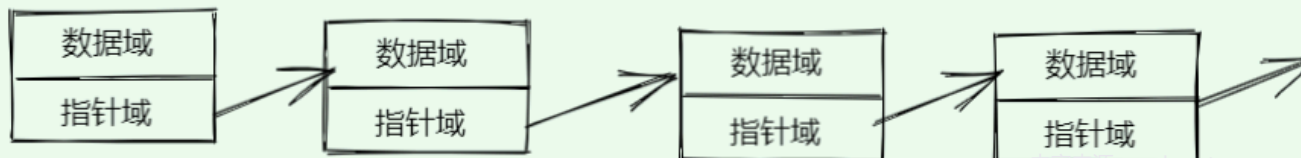
单向



循环



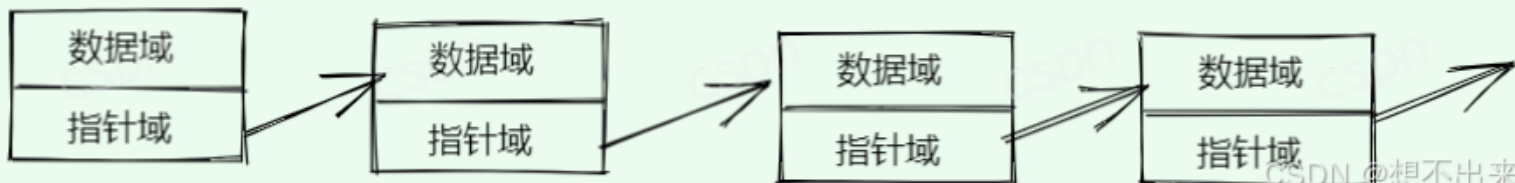
不循环



虽然有这么多的链表的结构，但是我们实际中最常用还是两种结构：

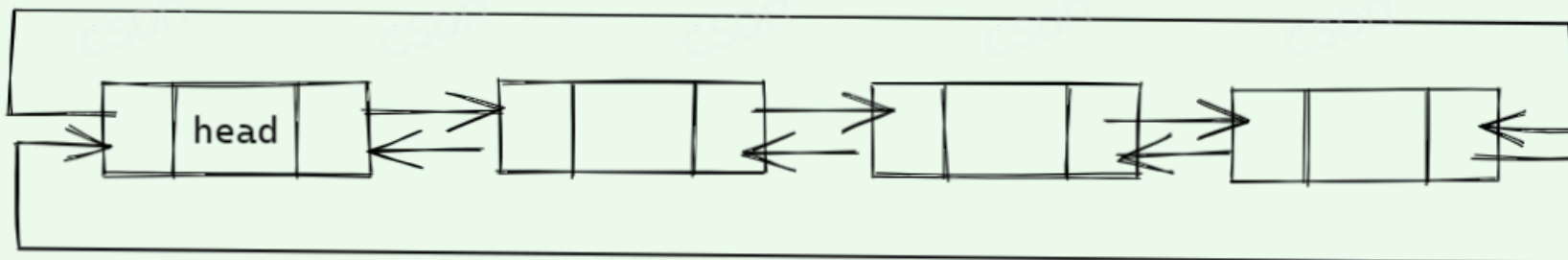
CSDN @想不出来起什么名字  
原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>  
作者主页: <https://blog.csdn.net/ting1fengyu1>

不带头单向不循环链表：即**单链表**



带头双向循环链表：即**双向链表**

双向链表



单链表：结构简单，一般不会单独用来存数据。实际中更多是作为其他数据结构的子结构

双向链表：结构最复杂，一般用在单独存储数据。实际中使用的 **链表数据结构**，都是带头双向循环链表

## 单链表的实现

定义一个单链表

```
1 //不带头不循环单向链表
2 typedef int SLDataType;
3 typedef struct SListNdoe {
4     SLDataType data;//数据域
5     struct Listnode* next;//指针域
6 }SLTNode;
```

代码解读

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

## 定义接口

```
1 //链表的销毁
2 void SLTDestory(SLTNode** pplist);
3
4 //插入操作
5 void SLTPushHead(SLTNode** pplist, SLDataType input);
6 void SLTPushBack(SLTNode** pplist, SLDataType input);
7 //指定位置之前插入元素
8 void SLTPushPosFront(SLTNode** pplist, SLTNode* pos,SLDataType input);
9 //指定位置之后插入元素
10 void SLTPushPosAfter(SLTNode* pos,SLDataType input);
11
12 //删除操作
13 void SLTPopBack(SLTNode** pplist);
14 void SLTPoplist(SLTNode** pplist);
15 //删除指定位置之后的一个元素
16 void SLTPopPosAfter(SLTNode* pos);
17 //删除指定位置元素
18 void SLTPopPos(SLTNode** pplist, SLTNode* pos);
19
20 //查找元素
21 SLTNode* SLTFind(SLTNode* plist, SLDataType target);
22
23 //链表打印
24 void SLTPrint(SLTNode* plist);
25
26 //创建新的节点
27 SLTNode* SLTBuyNode(SLDataType input);
```



代码解读

## 插入操作

在插入节点之前需要定义新的接口，开辟一个节点的空间

```
1 SLTNode* SLTBuyNode(SLDataType input) {
```

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

2 | SLTNode* newNode = (SLTNode*)malloc(sizeof(SLTNode));
3 | if (newNode == NULL) {
4 |     perror("malloc() err!");
5 |     return;
6 | }
7 | newNode->data = input;
8 | newNode->next = NULL;
9 |
10 | return newNode;
11 | }

```

代码解读

## 头插

```

1 | void SLTPushHead(SLTNode** pplist, SLDataType input) {
2 |     //检查二级指针是否有效
3 |     assert(pplist);
4 |
5 |     SLTNode* newNode = SLTBuyNode(input);
6 |
7 |     newNode->next = *pplist;
8 |     *pplist = newNode;
9 | }

```

代码解读

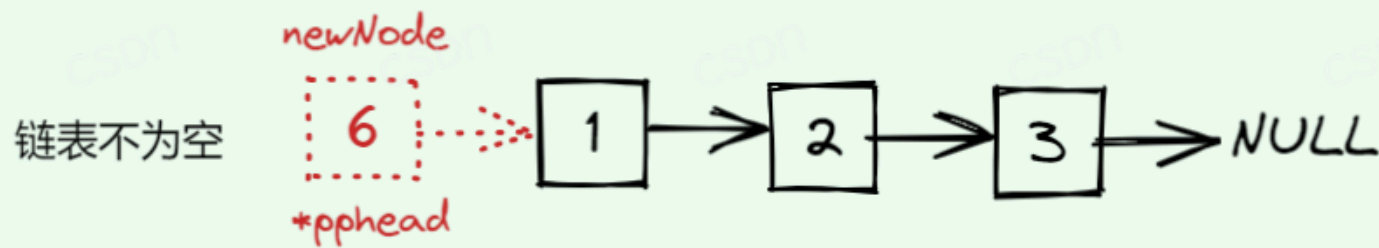
内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

# 头插 6



```
newNode->next = *pphead;  
*pphead = newNode;
```

CSDN @想不出来起什么名字

为什么要定义二级指针?

如果函数参数定义成一级指针, 那么应改成SLTPushHead(plist, 1)了, 这时候就是传值操作, SLTPushHead()函数栈帧结束后, 值也随之销毁

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>





```
printf("\n");
SLTPushBack(NULL, 4);
SLTPrint(plist);
```

不能传空地址

```
return 0;
```

所以在操作之前需要断言



Debug Error!

Program:

D:\CODE\DataStructure\DataStructureClone\SList\Debug\SList.exe

abort() has been called

(Press Retry to debug the application)

中止(A)

重试(R)

忽略(I)

CSDN @想不出来起什么名字

## 尾插

```
1 //尾插
2 void SLTPushBack(SLTNode** pplist, SLDataType input) {
3     //检查二级指针是否有效
4     assert(pplist);
5
6     SLTNode* newNode = SLTBuyNode(input);
7     SLTNode* ptail = *pplist;
8     //链表为空 新节点就是尾节点
9     if (*pplist == NULL) {
10         ptail = newNode;
11         return;
12     }
13     //链表不为空 找尾节点
14     while (ptail->next) {
15         ptail = ptail->next;
16     }
17     ptail->next = newNode;
18 }
```

代码解读

内容来源: csdn.net

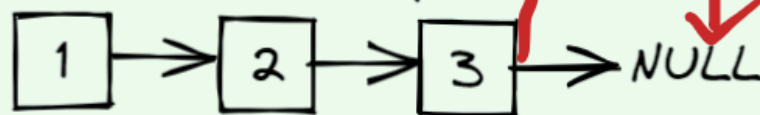
作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

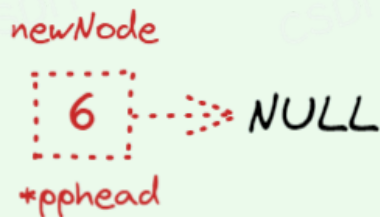
## 尾插 6

链表不为空 先找尾



找到尾结点的条件  
while(ptail->next)

链表为空



ptail->next = newNode;

CSDN @想不出来起什么名字

指定位置需要查找 查找指定节点

```
1
2 SLTNode* SLTFind(SLTNode* plist, SLDataType target) {
3     assert(plist);
4     SLTNode* pcur = plist;
5     while (pcur) {
6         if (pcur->data == target) {
7             return pcur;
8         }
9         pcur = pcur->next;
10    }
11    return NULL;
12 }
```

代码解读

指定位置之前插入节点

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

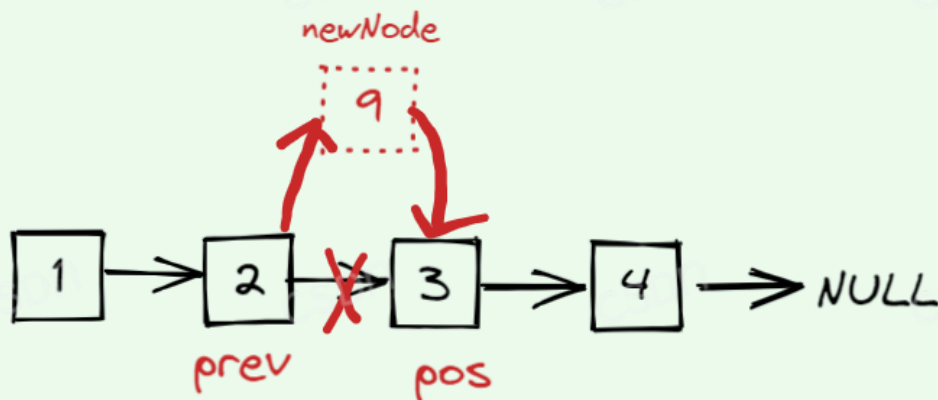
1 //指定位置之前插入元素
2
3 void SLTPushPosFront(SLTNode** pplist, SLTNode* pos, SLDataType input) { 3 | assert(pplist);
4     assert(pos);
5     assert(*pplist);
6
7     //pos刚好是第一个节点 头插
8     if (pos == *pplist) {
9         SLTPushHead(pplist, input);
10        return;
11    }
12
13    //pos不是第一个节点 找pos的前驱节点
14    SLTNode* newNode = SLTBuyNode(input);
15    SLTNode* prev = *pplist;
16    while (prev != NULL && prev->next != pos) {
17        prev = prev->next;
18    }
19    //添加判断, 看是否找到了pos的前驱节点
20    if (prev == NULL) {
21        //说明没找到pos, 可能pos不属于该链表, 可以根据具体需求进行错误处理, 这里简单释放刚申请的节点并返回
22        free(newNode);
23        return;
24    }
25    newNode->next = pos;
26    prev->next = newNode;
27 }

```


  
 代码解读

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)  
 作者昵称: 想不出来起什么名字  
 原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>  
 作者主页: <https://blog.csdn.net/ting1fengyu1>

在指定位置之前插入元素  $pos = 3$   $input = 9$



关键:

$prev$   $newNode$   $pos$  三者连接顺序

如果 $pos$ 刚好是第一个节点, 就头插即可

$newNode \rightarrow next = pos;$   
 $prev \rightarrow next = newNode;$

CSDN @想不出来起什么名字

## 指定位置之后插入节点

一定要注意指针链接顺序!!!!

```
1 //指定位置之后插入元素
2 void SLTPushPosAfter(SLTNode* pos, SLDataType input) {
3     //只需要找到pos的位置即可 pos之前的节点无需考虑
4     assert(pos);
5     SLTNode* newNode = SLTBuyNode(input);
6
7     newNode->next = pos->next;
8     pos->next = newNode;
9 }
```

代码解读

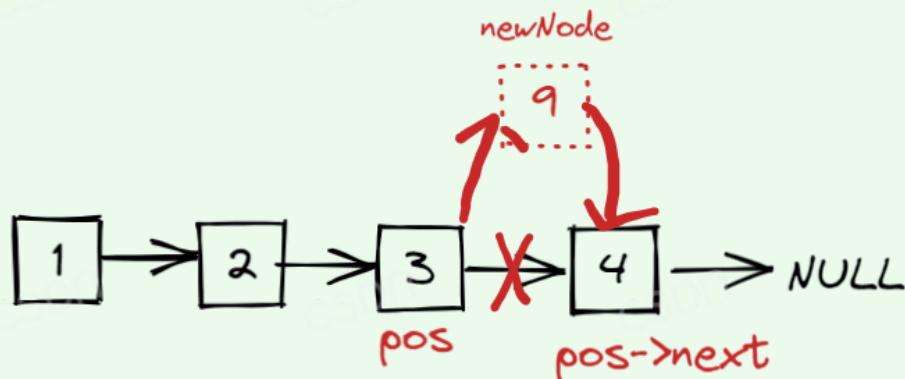
内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

在指定位置之后插入元素  $pos = 3$   $input = 9$



$newNode \rightarrow next = pos \rightarrow next;$   
 $pos \rightarrow next = newNode;$

~~$pos \rightarrow next = newNode;$   
 $newNode \rightarrow next = pos \rightarrow next;$~~

pos的next指向了新节点  
而新节点的next指向pos->next是newNode本身

CSDN @想不出来起什么名字

## 删除操作

### 头删

```
1 //头删
2 void SLTPoplist(SLTNode** pplist) {
3     //检查二级指针是否有效
4     assert(pplist);
5     //链表不能为空 空链表不能执行删除操作
6     assert(*pplist);
7     //把第二个节点当作第一个节点 释放第一个节点
8     SLTNode* next = (*pplist)->next;
9     free(*pplist);
10    *pplist = next;
```

内容来源: csdn.net

作者昵称: 想不出来起什么名字

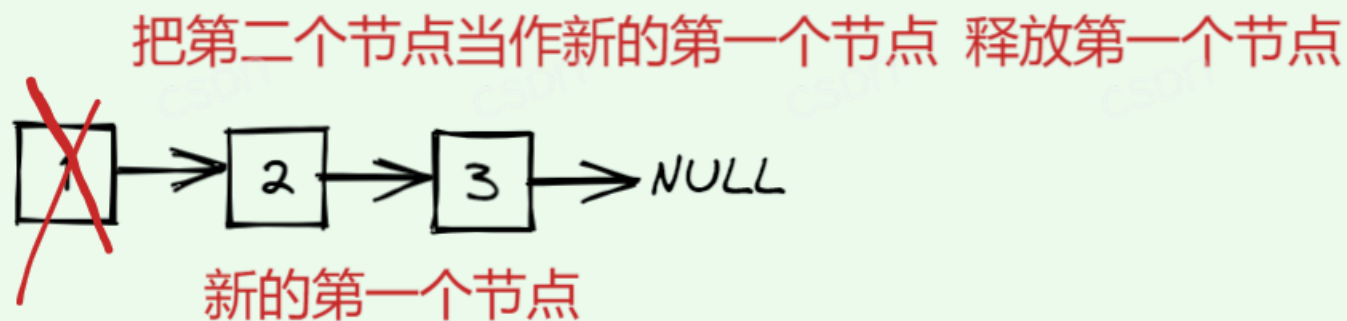
原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

## 头删

链表为空 不能删除

链表不为空



CSDN @想不出来起什么名字

## 尾删

```
1 //尾删
2 void SLTPopBack(SLTNode** pplist) {
3     //检查二级指针是否有效
4     assert(pplist);
5     //链表不能为空 空链表不能执行删除操作
6     assert(*pplist);
7
8     //链表不为空
9     // 只有一个节点
10    if ((*pplist)->next == NULL) {
11        free(*pplist);
12        *pplist = NULL;
13        return;
14    }
15    //有多个节点
```

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```
16 //找尾 保存前驱节点
17 SLTNode* ptail = *pplist;
18 SLTNode* prev = NULL;
19 while (ptail->next) {
20     prev = ptail;
21     ptail = ptail->next;
22 }
23 //销毁尾节点
24 prev->next = NULL;
25 free(ptail);
26 ptail = NULL;
27 }
```



代码解读

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

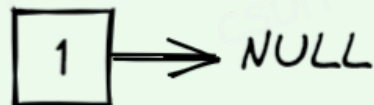
作者主页: <https://blog.csdn.net/ting1fengyu1>

## 尾删

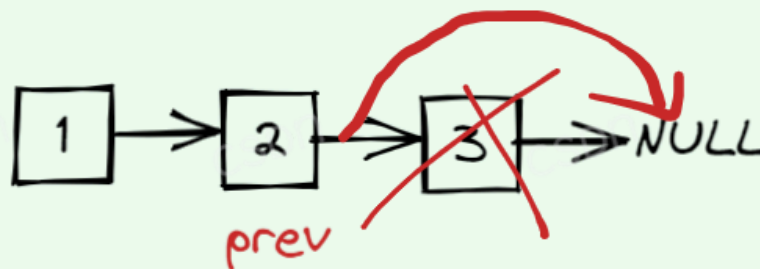
链表为空 不能删除

链表不为空

链表只有一个节点



链表有多个节点



要保存尾节点前的前驱节点

## 删除指定位置节点

```
1 //删除指定位置元素
2 void SLTPopPos(SLTNode** pplist, SLTNode* pos) {
3     assert(pplist);
4     assert(pos);
5     assert(*pplist);
6
7     //pos刚好在第一个节点位置
8     if (pos == *pplist) {
9         //头删
```

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>



```
10     SLTPoplist(pplist);11     return;
12 }
13 SLTNode* prev = *pplist;
14 while (prev != NULL && prev->next != pos) {
15     prev = prev->next;
16 }
17 prev->next = pos->next;
18 free(pos);
19 pos = NULL;
```



代码解读

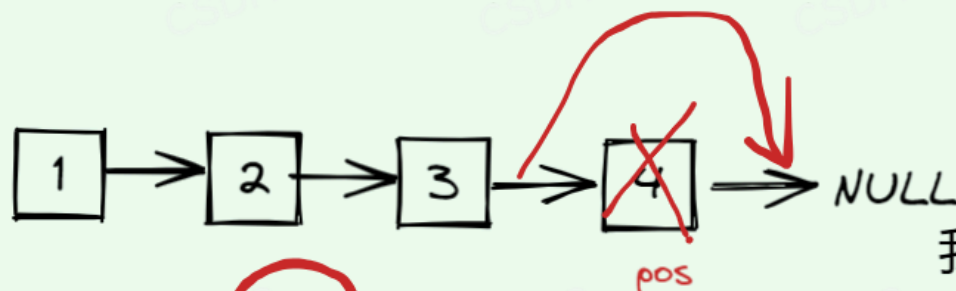
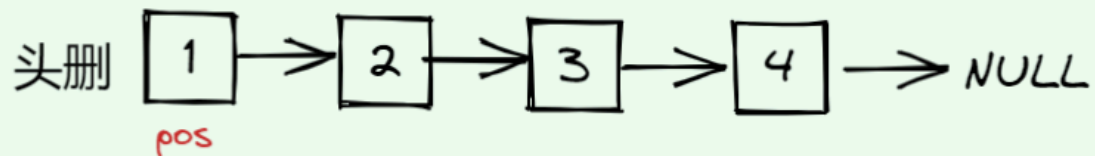
内容来源: csdn.net

作者昵称: 想不出来起什么名字

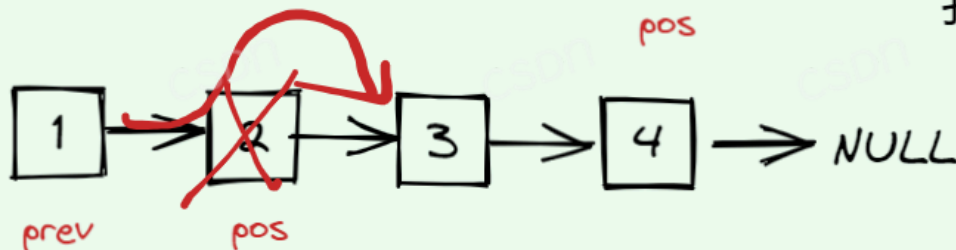
原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

## 删除pos位置的节点



我们不清楚pos是否为尾节点



与这种情况归为一类

CSDN @想不出来起什么名字

## 删除指定位置之后的节点

```
1 //删除指定位置之后的一个元素
2 void SLTPopPosAfter(SLTNode* pos) {
3     assert(pos);
4     assert(pos->next);
5
6     SLTNode* del = pos->next;
7     pos->next = del->next;
8     free(del);
```

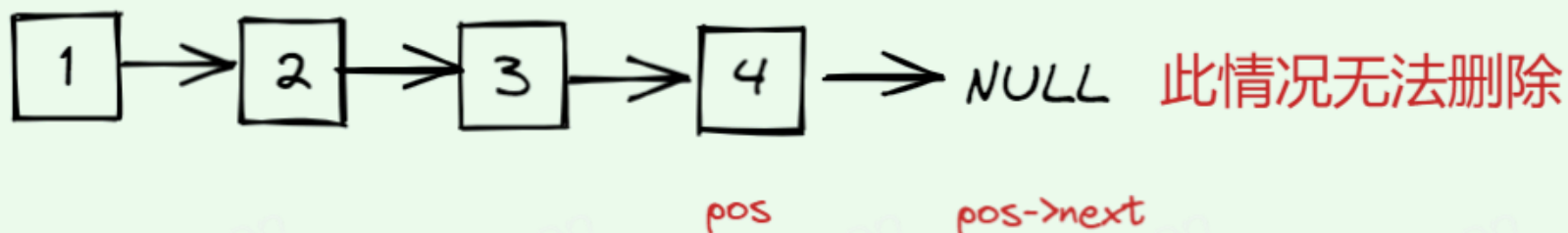
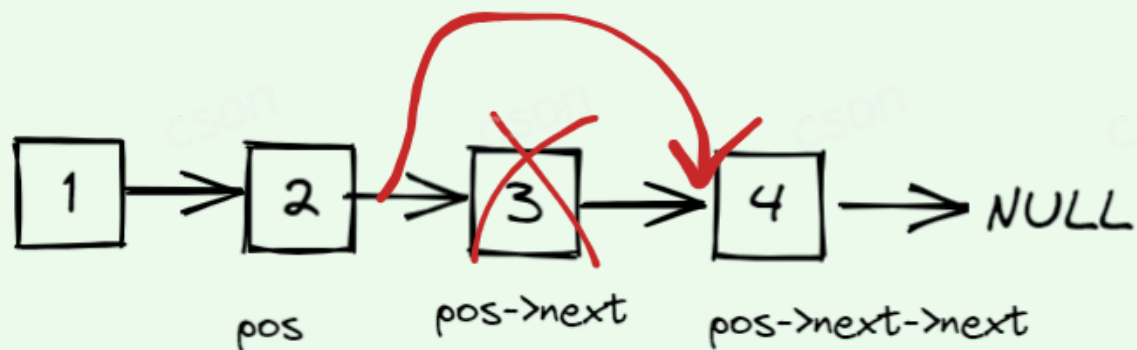
内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

删除指定位置之后的一个元素  $pos = 2$



CSDN @想不出来起什么名字

## 其他操作

### 销毁链表

```
1 //链表的销毁
2 void SLTDestory(SLTNode** pplist) {
```

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

3      assert(pplist); 4      assert(*pplist);
5
6      SLTNode* del = *pplist;
7      while (del) {
8          SLTNode* pcur = del;
9          free(pcur);
10         del = del->next;
11     }
12     *pplist = NULL;
13 }

```

代码解读

## 打印链表

```

1 void SLTPrint(SLTNode* plist) {
2     SLTNode* pcur = plist;
3     while (pcur)
4     {
5         printf("%d->", pcur->data);
6         pcur = pcur->next;
7     }
8     printf("NULL\n");
9 }

```

代码解读

## 双向链表

### 相关定义

先定义一个双向链表：

```

typedef int LTDataType;
typedef struct List {
    LTDataType data;

```

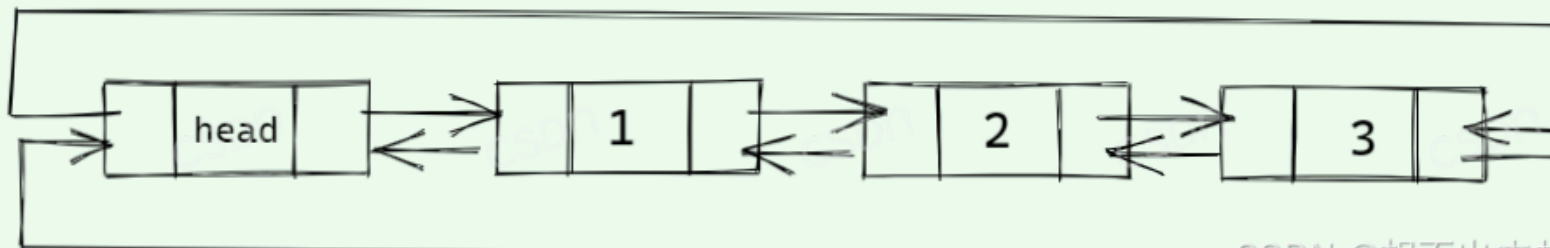
内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

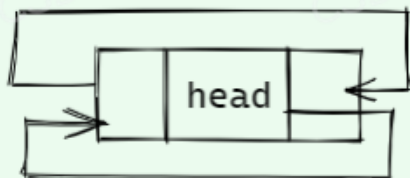
作者主页: <https://blog.csdn.net/ting1fengyu1>

```
struct List* next;//指向后继节点
struct List* prev;//指向前驱节点
}LTNode;
```



CSDN @想不出来起什么名字

双向链表为空的情况——



只有一个哨兵位，不存储有效数据

CSDN @想不出来起什么名字

## 定义接口

参数设计，要一级指针还是二级指针？

一级指针，因为我们现在有一个哨兵位可以找到链表所有节点

二级指针是为了解引用一次操作链表，没有哨兵位，就需要解引用二级指针来操作链表

而有了哨兵位，有了天然的可操作链表的优势

```
1 //创建新的节点
2 ListNode* ListNewNode(LTDataType input);
3 // 初始化链表
4 ListNode* ListInit();
```

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

5 // 双向链表销毁
6 void ListDestory(ListNode* plist);
7 // 双向链表打印
8 void ListPrint(ListNode* plist);
9 // 双向链表尾插
10 void ListPushBack(ListNode* plist, LTDataType input);
11 // 双向链表尾删
12 void ListPopBack(ListNode* plist);
13 // 双向链表头插
14 void ListPushFront(ListNode* plist, LTDataType input);
15 // 双向链表头删
16 void ListPopFront(ListNode* plist);
17 // 双向链表查找
18 ListNode* ListFind(ListNode* plist, LTDataType input);
19 // 双向链表在pos的前面进行插入
20 void ListInsert(ListNode* pos, LTDataType iinput);
21 // 双向链表删除pos位置的结点
22 void ListErase(ListNode* pos);

```



代码解读

## 插入操作

### 尾插

```

1 void ListPushBack(ListNode* plist, LTDataType input) {
2     assert(plist);
3     ListNode* newNode = ListNewNode(input);
4     ListNode* pcur = plist->next;
5
6     newNode->next = plist;
7     newNode->prev = plist->prev;
8
9     plist->prev->next = newNode;
10    plist->prev = newNode;
11

```

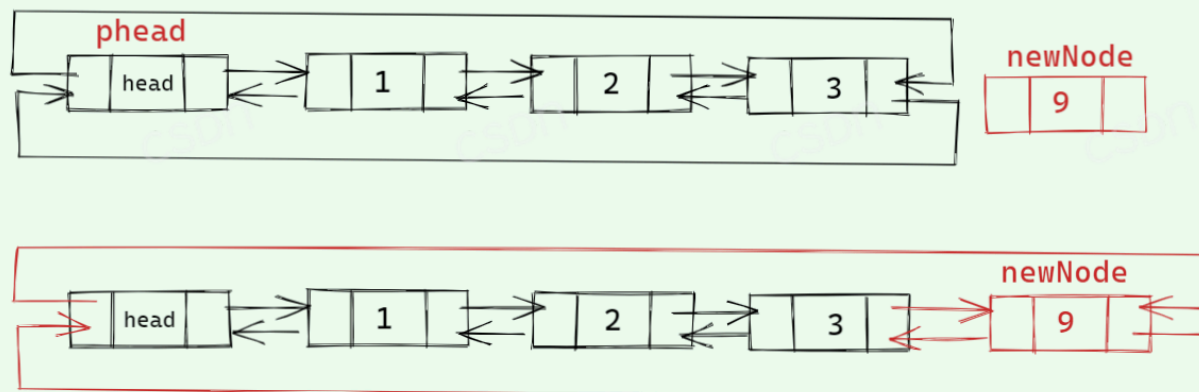
内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

尾插9



```
newNode->next = phead;
newNode->prev = phead->prev;
phead->prev->next = newNode;
phead->prev = newNode;
```

CSDN @想不出来起什么名字

头插

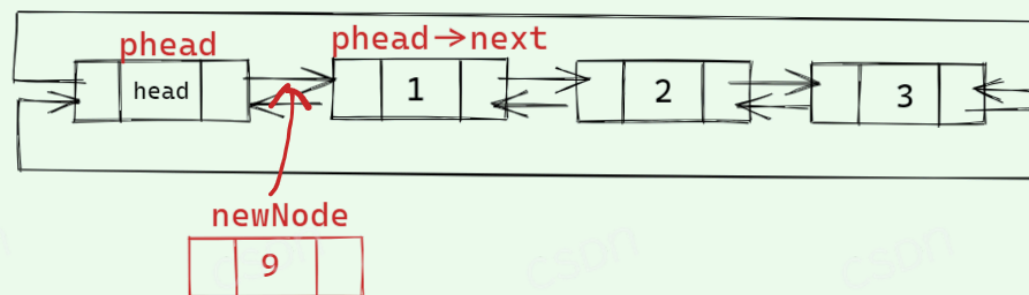
```
1 // 双向链表头插
2 void ListPushFront(ListNode* plist, LTDataType input) {
3     assert(plist);
4
5     ListNode* newNode = ListNewNode(input);
6
7     newNode->next = plist->next;
8     newNode->prev = plist;
9
10    plist->next->prev = newNode;
11    plist->next = newNode;
12 }
```

内容来源: csdn.net

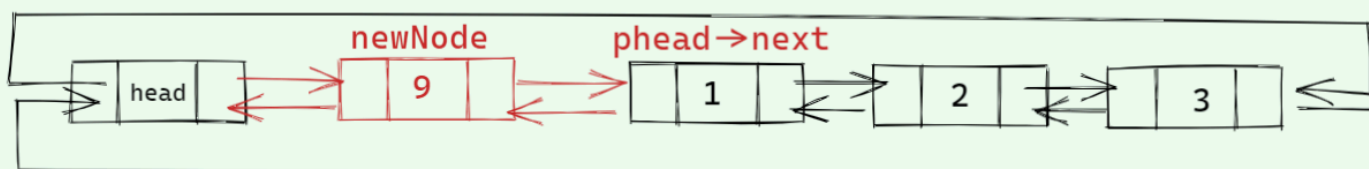
作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>作者主页: <https://blog.csdn.net/ting1fengyu1>

## 头插9 在哨兵位之后插入新节点



```
newNode->next = phead->next;
newNode->prev = phead;
phead->next->prev = newNode;
phead->next = newNode;
```



CSDN @想不出来起什么名字

## 指定位置之前插入（先要找到pos的位置）

```
1 // 双向链表在pos的前面进行插入
2 void ListInsert(ListNode* pos, LTDataType input) {
3     assert(pos);
4
5     ListNode* newNode = ListNewNode(input);
6
7     newNode->next = pos;
8     newNode->prev = pos->prev;
9
10    pos->prev->next = newNode; // 这两个顺序不能错
11    pos->prev = newNode;
12 }
```

代码解读

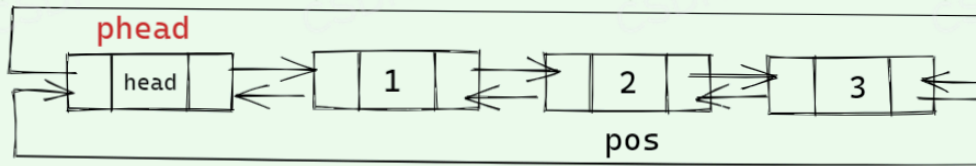
内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>作者主页: <https://blog.csdn.net/ting1fengyu1>

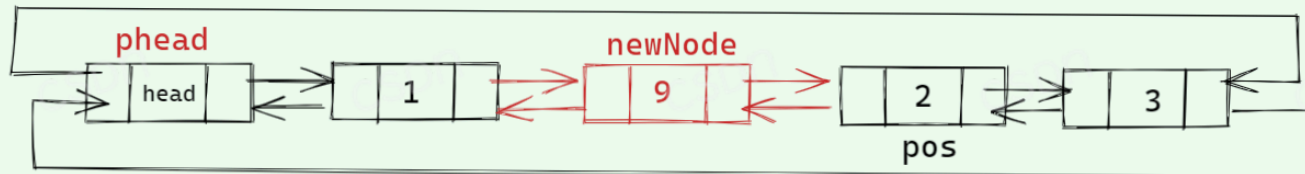


在pos之前插入9



```
newNode->next = pos;  
newNode->prev = pos->prev;
```

```
pos->prev->next = newNode; //这两个顺序不能错  
pos->prev = newNode;
```



CSDN @想不出来起什么名字

## 删除操作

### 头删

```
1 void ListPopFront(ListNode* plist) {  
2     assert(plist->next != plist); //链表不能为空  
3  
4     ListNode* del = plist->next;  
5     ListNode* next = del->next;  
6  
7     next->prev = plist;  
8     plist->next = next; //顺序不能错  
9  
10    free(del);  
11    del = NULL;  
12 }
```

代码解读

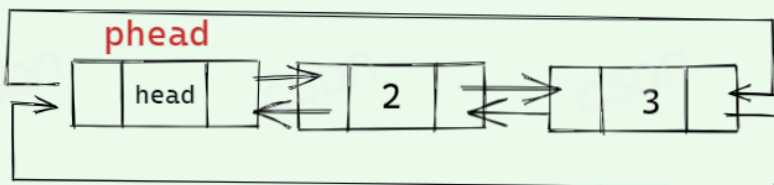
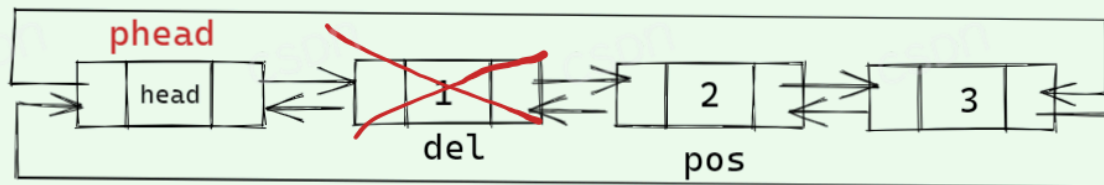
内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

头删



```
ListNode* del = plist->next;  
ListNode* next = del->next;
```

```
next->prev = plist;  
plist->next = next; //顺序不能错
```

CSDN @想不出来起什么名字

尾删

```
1 // 双向链表尾删  
2 void ListPopBack(ListNode* plist) {  
3     assert(plist->prev != plist);  
4  
5     ListNode* del = plist->prev;  
6     ListNode* prev = del->prev;  
7  
8     prev->next = plist;  
9     plist->prev = prev;  
10  
11     free(del);  
12     del = NULL;  
13 }
```

代码解读

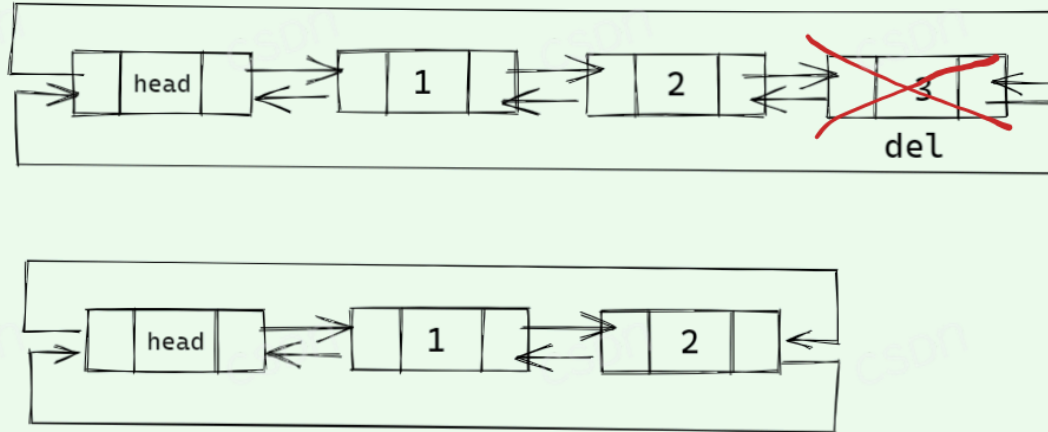
内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

## 尾删



```
ListNode* del = plist->prev;  
ListNode* prev = del->prev;
```

```
prev->next = plist;  
plist->prev = prev; //顺序不能错
```

CSDN @想不出来起什么名字

## 删除指定位置（先要找到pos的位置）

```
1 // 双向链表删除pos位置的结点  
2 void ListErase(ListNode* pos) {  
3     assert(pos);  
4  
5     pos->prev->next = pos->next;  
6     pos->next->prev = pos->prev;  
7  
8     free(pos);  
9     pos = NULL;  
10 }
```

代码解读

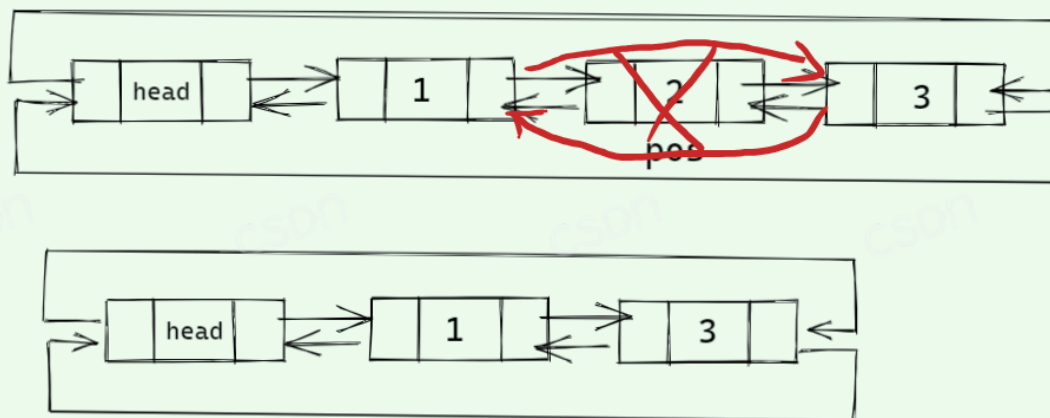
内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

## 删除指定位置



```
pos->prev->next = pos->next;  
pos->next->prev = pos->prev;
```

CSDN @想不出来起什么名字

## 其他操作

### 创建新节点

```
1 //创建新的节点  
2 ListNode* ListNewNode(LTDataType input) {  
3     ListNode* newNode = (ListNode*)malloc(sizeof(ListNode));  
4     if (newNode == NULL) {  
5         perror("malloc() err!");  
6         return 1;  
7     }  
8     newNode->data = input;  
9     newNode->next = newNode->prev = newNode;  
10 }
```

代码解读

### 查找节点

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```

1 // 双向链表查找
2 ListNode* ListFind(ListNode* plist, LTDataType target) {
3     assert(plist);
4
5     ListNode* pcur = plist->next;
6
7     while (pcur != plist) {
8         if (pcur->data == target) {
9             return pcur;
10        }
11        pcur = pcur->next;
12    }
13    return NULL;
14 }

```

代码解读

## 打印链表

```

1 // 双向链表打印
2 void ListPrint(ListNode* plist) {
3     assert(plist);
4
5     ListNode* pcur = plist->next;
6     while (pcur != plist) {
7         printf("%d->", pcur->data);
8         pcur = pcur->next;
9     }
10    printf("\n");
11 }

```

代码解读

## 销毁链表

```

1 // 双向链表销毁
2 void ListDestory(ListNode* plist) {
3     assert(plist);

```

内容来源: csdn.net

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

```
4 |
5 |     ListNode* pcur = plist->next;
6 |     while (pcur != plist) {
7 |         ListNode* next = pcur->next;
8 |         free(pcur);
9 |         pcur = next;
10 |    }
11 |    //只剩下哨兵位了
12 |    free(plist);
13 |    plist = NULL;
14 | }
```

代码解读

## 链表和顺序表的对比

内容来源: [csdn.net](https://blog.csdn.net/ting1fengyu1)

作者昵称: 想不出来起什么名字

原文链接: <https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页: <https://blog.csdn.net/ting1fengyu1>

	顺序表	链表
存储空间	数组，连续存储	结构体，不一定连续存储
是否支持随机访问	支持，访问效率： $O(1)$	不支持： $O(n)$
任意位置插入或删除节点	可能需要挪动数据，效率低 $O(n)$	只需修改指针指向
应用场景	元素高效存储+频繁访问	任意位置插入和删除频繁
缓存利用率	高	低

CSDN @想不出来起什么名字

## 总结

- 理解链表的现实意义
- 链表的具体分类
- 单链表的增删查改
- 双向链表的增删查改
- 顺序表和链表的区别具体应用场景

内容来源：csdn.net

作者昵称：想不出来起什么名字

原文链接：<https://blog.csdn.net/ting1fengyu1/article/details/144543335>

作者主页：<https://blog.csdn.net/ting1fengyu1>