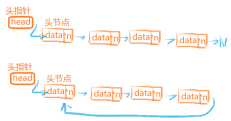
单向链表



末尾节点：

如果是循环链表则末尾节点的下一位是头节点（后继指针指向头节点/后继节点为头节点）

如果是非循环链表末尾节点的下一位是空（后继指针指向空）

节点设计：


```
// 节点设计
typedef struct list
{
    Data_Type Num ;   //数据
    struct list * Next ; // 后继指针
}List , *P_List;
```

初始化：

非循环：

循环：

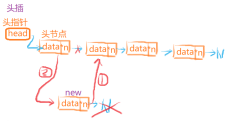

```
Node * new =  calloc(1 , sizeof(Node));
if(new == NULL)…
                                非循环
// 让结构体中 后继 指针指向空
new->Next = NULL ;

// 初始化节点的数据
new->data = data ;
```

```
                                循环
// 初始化新节点
P_List init_new_node(Data_Type data)
{
    P_List new = calloc(1, sizeof(List));

    new->Num = data;
    new->Next = new ;

    return new ; // 返回的是堆空间的内存地址
}
```

插入数据：

单向链表：

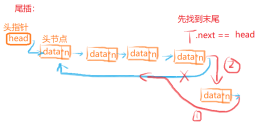
头插



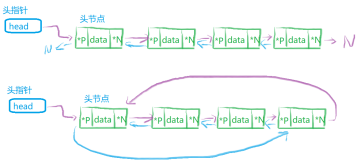
尾插：

单向循环链表：

尾插：



双向链表



末尾节点：

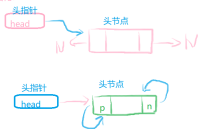
不循环链表：　末尾的下一个为空（后继指针指向空）

循环链表：　末尾节点的下一个为头节点（后继指针指向头节点 / 后继节点为头节点）

节点设计：


```
// 节点设计
typedef struct node
{
    Data_Type  data;        // 数据域
    struct node *prev ;     // 前驱指针
    struct node *next ;     // 后继指针
}Node , *P_Node ;
```

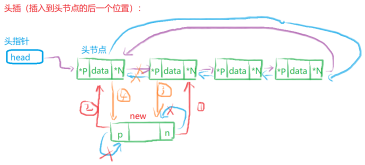
初始化：




```
P_Node new_node_init( Data_Type  data )
{
    P_Node new = calloc(1 , sizeof(Node));
    if( new == NULL )…

    new->data = data ;
    new->next = new->prev = new ;让双向循环链表的新节点的
                                  前后指针都指向自己
    return new ;
}
```

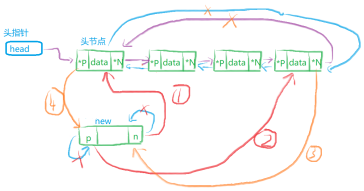
节点插入：

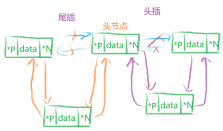
头插（插入到头节点的后一个位置）：



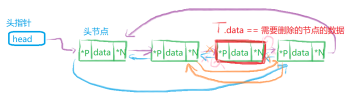
尾插：

插入到头节点的前面





节点删除：



内核链表

末尾节点：

末尾节点的小结构体中的后继指针指向头节点的小结构体

节点设计：


```
// 节点设计
typedef struct node
{
    Data_Type data ;
    struct list_head ptr ;
}Node , * P_Node;
```

```
/**
 * 在大结构体中定义一个小型结构体作为指针域
 */
struct list_head {
    struct list_head *next, *prev;
};
```

初始化：


```
P_Node kernelListInit(Data_Type data )
{
    // 申请大结构体的内存空间
    P_Node new = calloc(1, sizeof(Node));

    //初始化数据
    new->data = data ;

    // 初始化小结构体（指针）
    INIT_LIST_HEAD( &new->ptr ); 指针域（小结构体）的初始化由
                                 内核提供函数（头文件）
    return new ;
}
```

节点插入：


```
list_add( &new->ptr, &head->ptr );
```

```
next->prev = xnew;
xnew->next = next;
xnew->prev = prev;
prev->next = xnew;
```


```
__list_add(xnew, head, head->next);
```

```
static inline void __list_add(struct list_head *xnew,
                              struct list_head *prev,
                              struct list_head *next)
```

节点删除：


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

```
entry->next = LIST_POISON1;
entry->prev = LIST_POISON2;
```

```
list_del(struct list_head *entry)
```

```
__list_del(entry->prev, entry->next);
```

```
__list_del(struct list_head * prev, struct list_head * next)
```

```
#define LIST_POISON1  ((struct list_head *) 0x00100100)
#define LIST_POISON2  ((struct list_head *) 0x00200200)
```