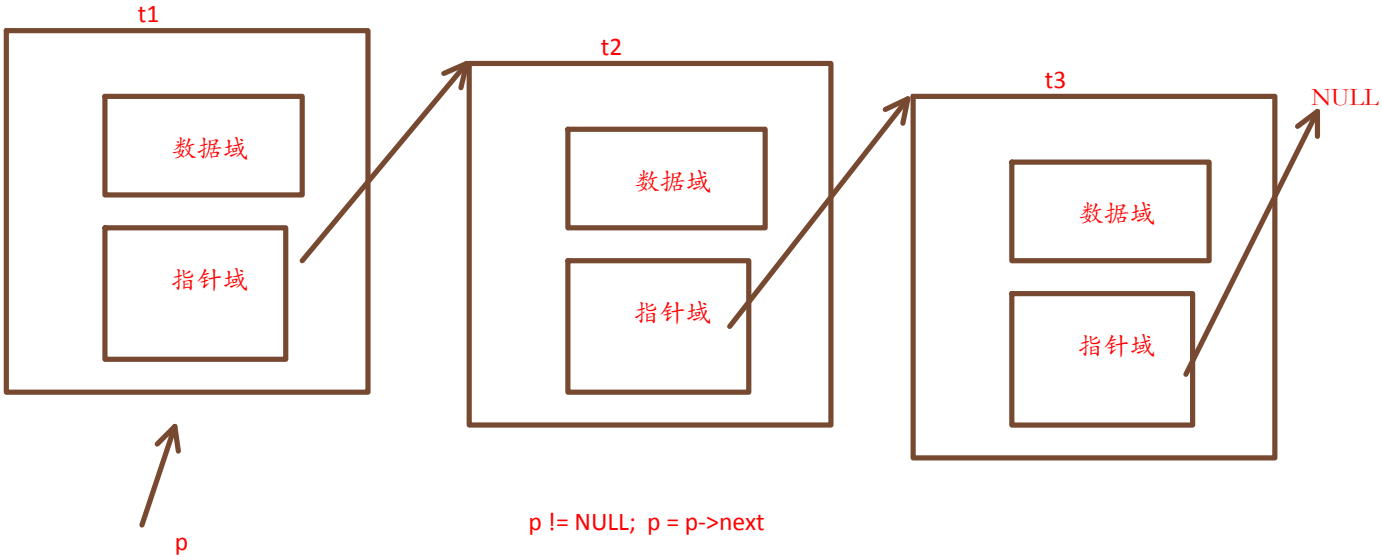
1 链表的节点

2015年11月21日 8:41

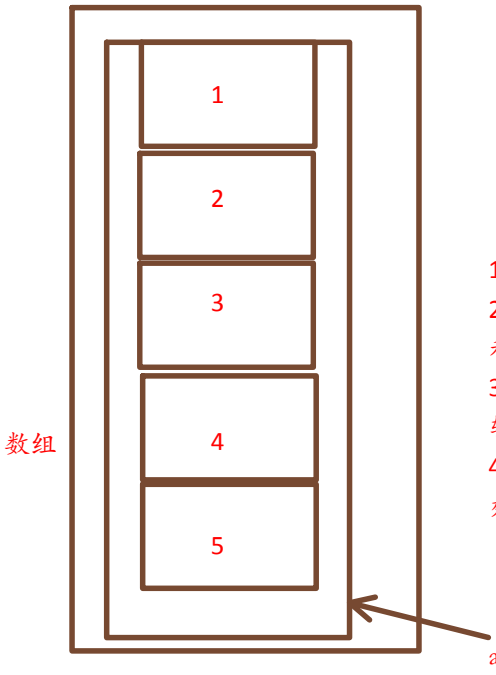
```
struct teacher
{
    int id; //数据域
    struct teacher *next; //指针域
}
```

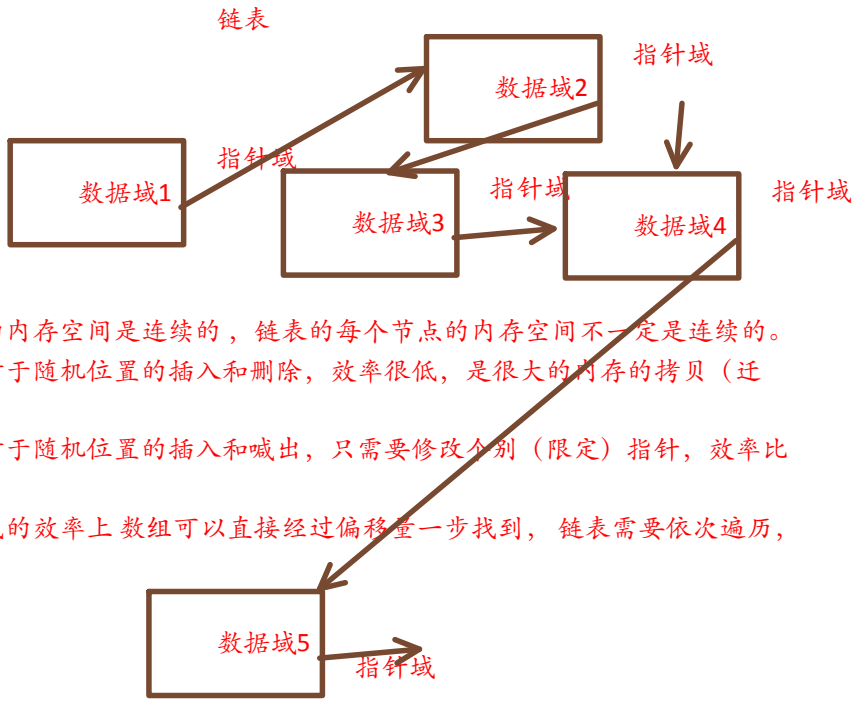


p != NULL; p = p->next

2 链表和数组的区别

2015年11月21日 9:33





1 数组的内存空间是连续的，链表的每个节点的内存空间不一定是连续的。

2 数组对于随机位置的插入和删除，效率很低，是很大的内存的拷贝（迁移）

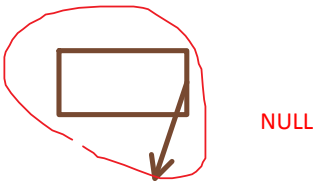
3 链表对于随机位置的插入和喊出，只需要修改个别（限定）指针，效率比较高

4 在查找的效率上 数组可以直接经过偏移量一步找到， 链表需要依次遍历，效率低

3 动态链表的创建

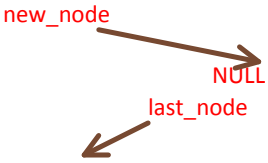
2015年11月21日 9:50

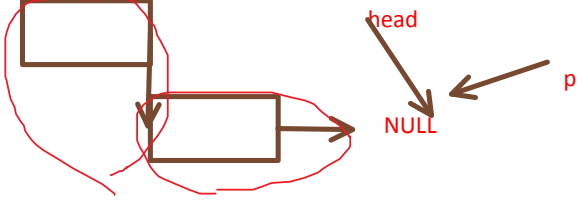
struct teacher *



p = head; p!=NULL;p=p->next

last_node->next = new_node;
last_node = new_node;



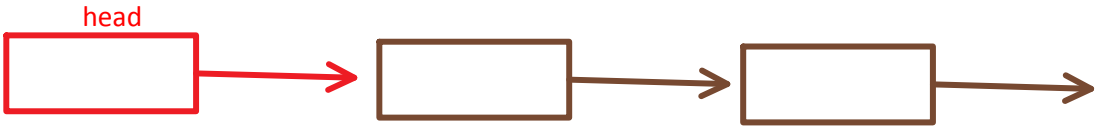


4 单向链表的带头与不带头

2015年11月21日 10:42

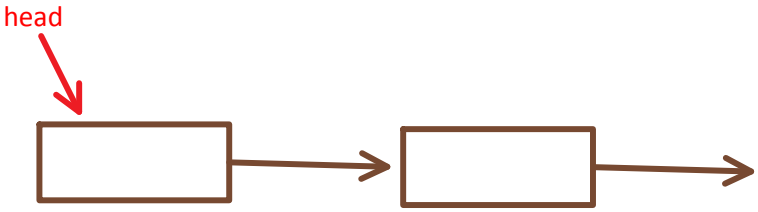
struct teacher head;

带头链表



struct teacher *head ;

不带头的链表



5 无头链表的插入方式

2015年11月21日 10:53

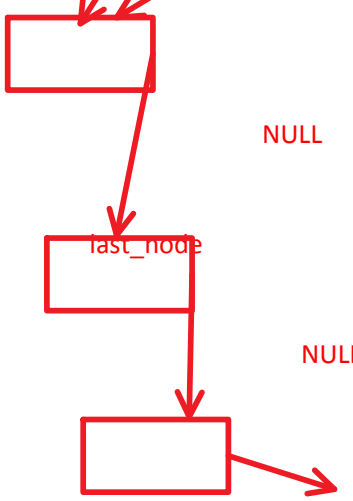
head

p = head

free(p);

head ==NULL:

head = new_node;



if (last_node->next == NULL)

last_node = last_node->next

last_node 是最后一个节点

从头插入

new_node = head;

6 无头链表的插入

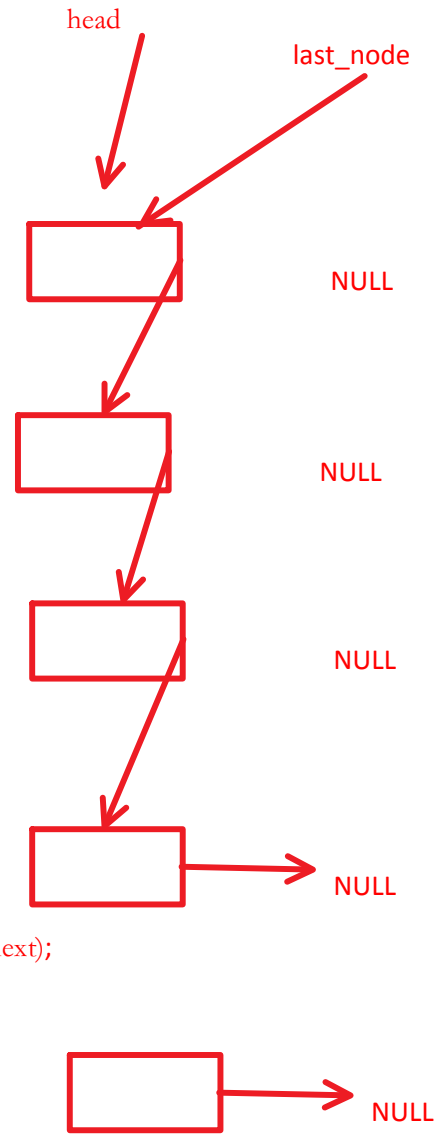
2015年11月21日 11:26

last_node = last_node->next
给last_node 向后偏移

```
int insert_node_to_end(struct node *new_node, struct node **head_p)
{
    struct node *head = NULL;
    struct node *last_node = NULL;

    if (new_node == NULL  || head_p == NULL) {
        return 0;
    }
    head = *head_p;

    if (head == NULL) {
        //链表此时是空链表
        head = new_node;
        //无头链表 的特点需要对head是否为空 进行判断
    }
    else {
        //找到这个last_node
        //last_node->next = new_node;


        for (last_node = head; last_node->next != NULL; last_node = last_node->next);

        //此时last_node 就是 最后一个节点地址
        last_node->next = new_node;
    }

    *head_p = head;

    return 0;
}
```

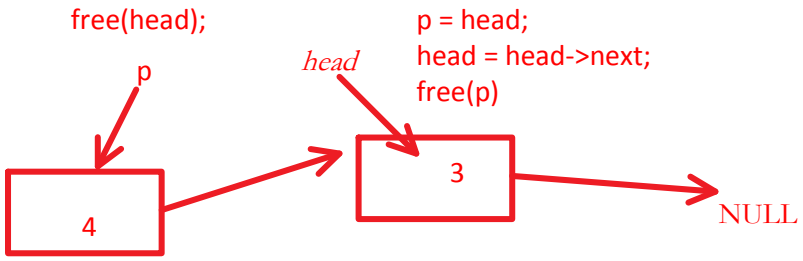
last_node = head
if (last_node->next == ULL) last_node 此时指向的节点是最后一个节点

last_node->next
= new_node



7 无头链表从头部加入

2015年11月21日 11:34



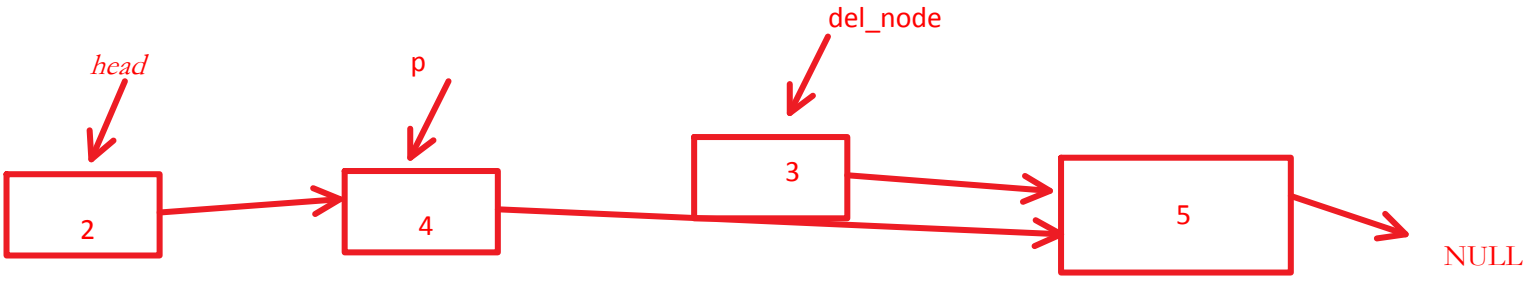

```
if（head == NULL)
{
	head = new_node;
}
else {
	new_node->next = head;
	head = new_node;
}
```

NULL

NULL

8 删除一个节点

2015年11月21日 14:51



if (删除的节点 ==head)

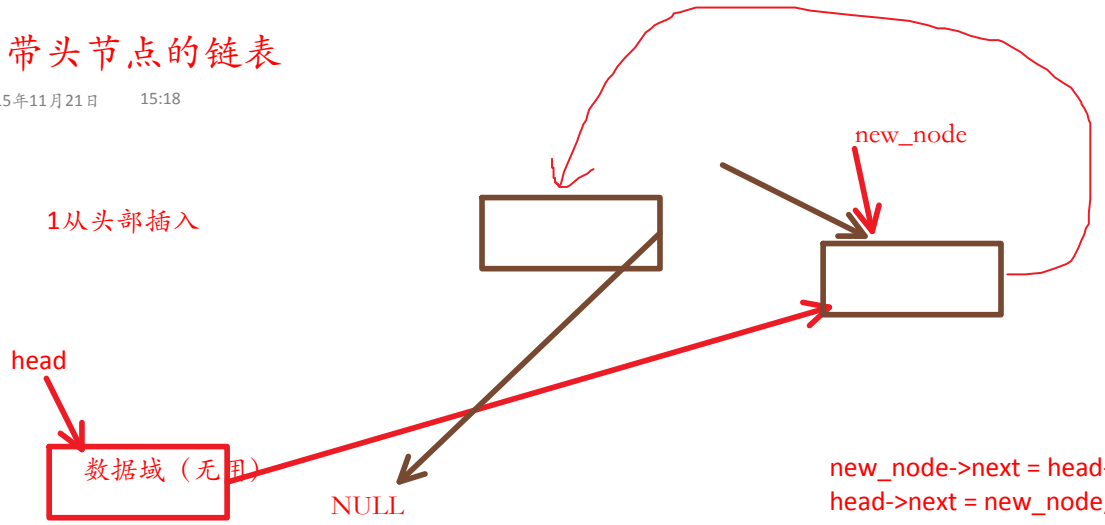
p->next == 3
p->next =p->next->next
free(temp_p);

NULL

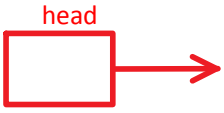
9 带头节点的链表

2015年11月21日 15:18

1从头部插入



new_node->next = head->next;
head->next = new_node;

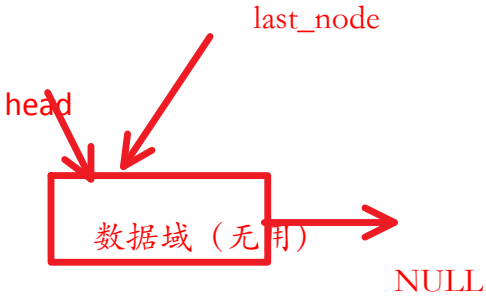


NULL

p = head; p->next != NULL
p =head->next; p != NULL

10 从尾部插入

2015年11月21日 16:00



last_node ->next = new_node;

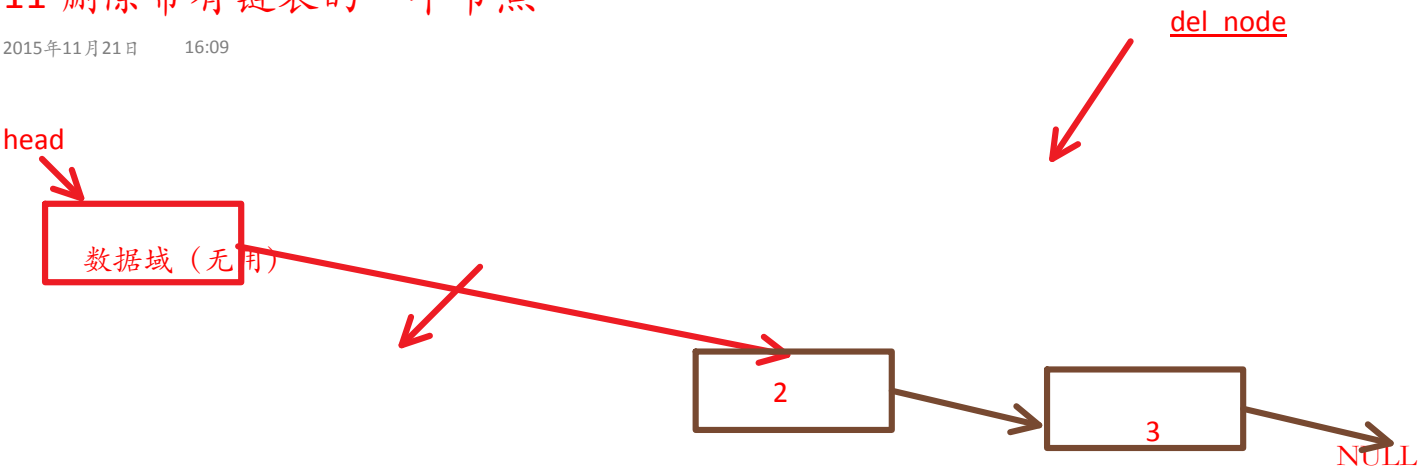
```
for (last_node = head; last_node->next != NULL; last_node=last_node->next)
{

}
```

last_node ->next ! = NULL;

last_node->next ==NULL --> last_node 就是最后一个节点

11 删除带有链表的一个节点

2015年11月21日 16:09



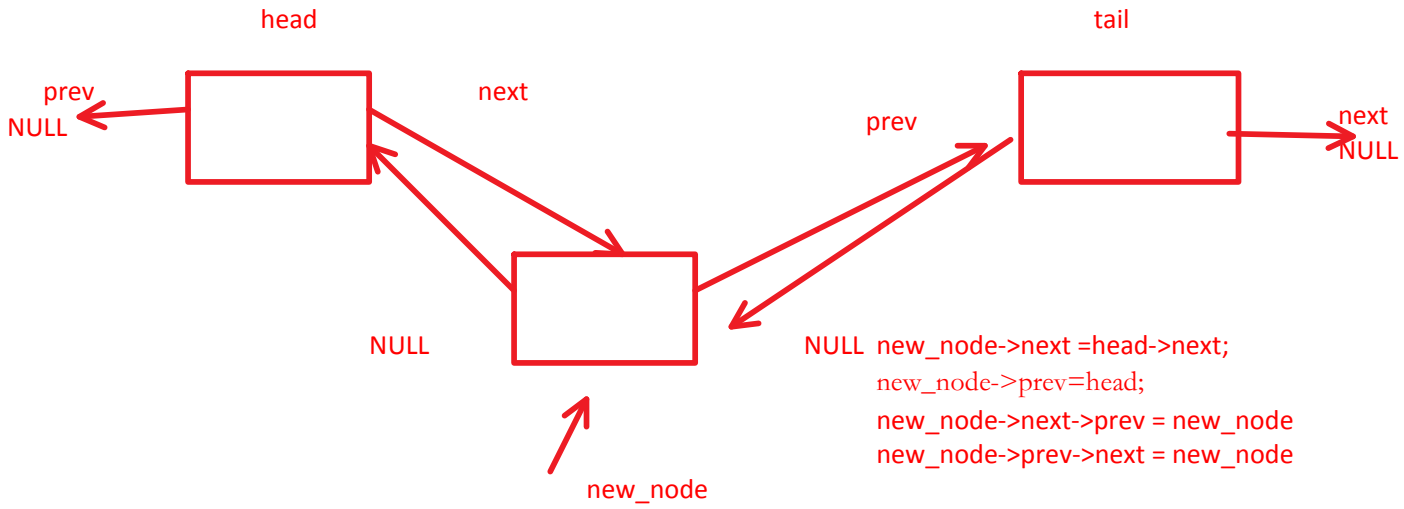

```
head->next = p->next;
free(p);
```

```
for（p = head; p->next != NULL; p = p->next)
{
        if (p->next == del_node) {
          找到了要删除的酒店就是 p->next
              p->next = p->next->next;
              free_node(del_node)
              break;
        }
}

for ( p =head->next; p != NULL; p = p->next) {
}
```

12 双向链表（带头）

2015年11月21日　　16:38

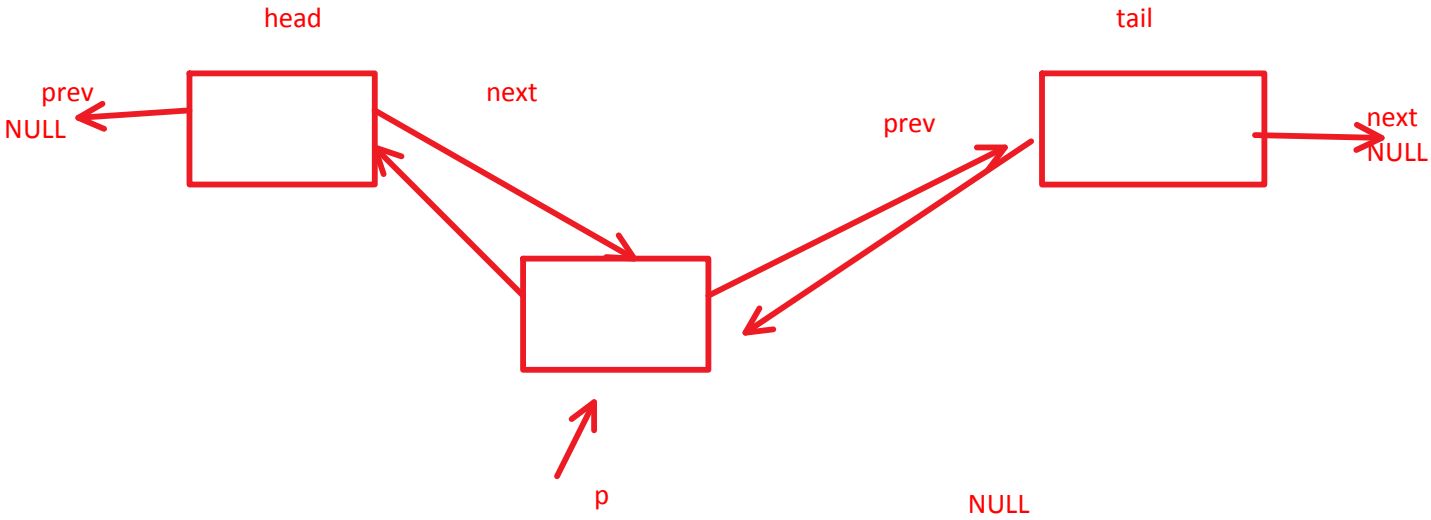
插入哪个节点
先改变这个被插入的 next 和prev



13 遍历一个双向链表

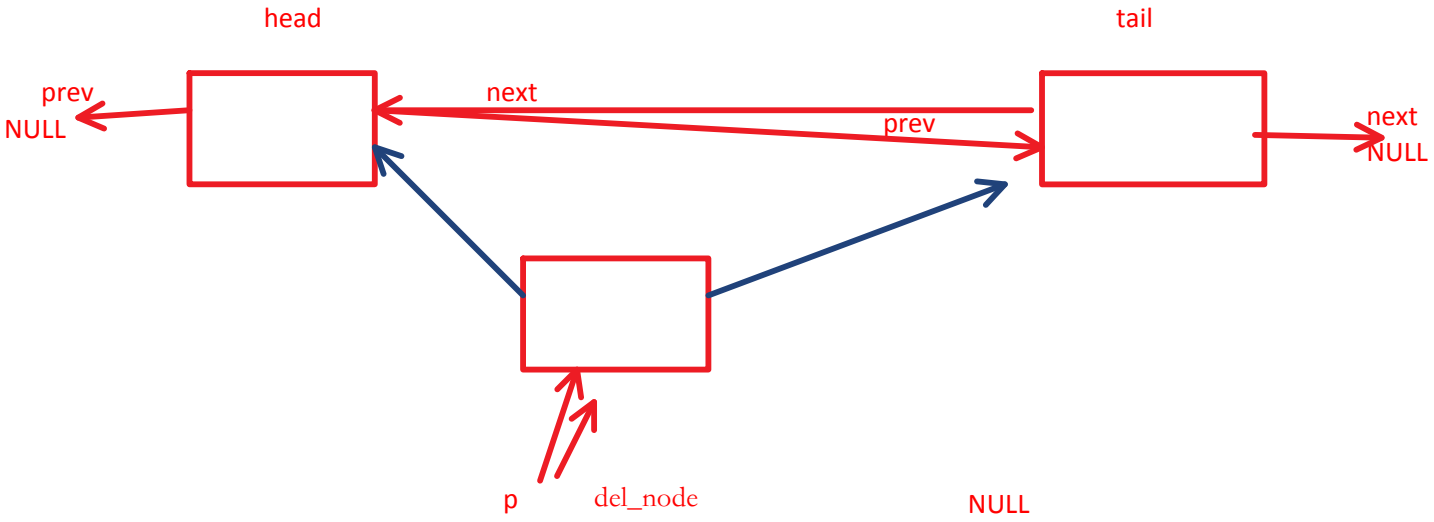
2015年11月21日　　16:54

p =head; p->next != tail;p = p->next
p =tail; p->prev != head; p = p->prev



13 双向链表删除一个节点

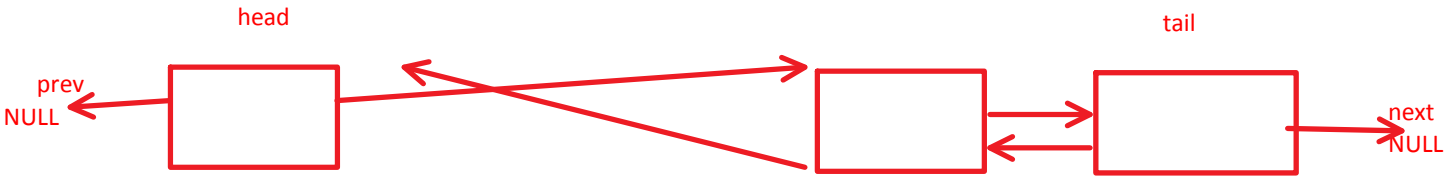
2015年11月21日 17:13




```
for (p = head->next; p != tail; p=p->next) {
      if (p == del_node) {
      //找到了要删除的节点就是p
        p->next->prev = p->prev
       p->prev->next = p->next
            free(p);

      }
}
```

14 销毁一个双向链表

2015年11月21日 17:22



next

p=head->next;

prev

NULL