递归翻转链表

> 给定一个单链表，翻转从第m个到第n个节点。

分析：

1. 对翻转整个链表可以采用以下算法：

```
 ListNode* reverseLinkLis(ListNode* head) {
  ListNode *dummy = new ListNode();
  while(head) {
    ListNode *p = head;
    head = head->next;
    p->next = dummy->next;
    dummy->next = p;
  }

  head = dummy->next;
  delete dummy;
  return head;
}
```

2. 翻转链表的前k个节点

```
 ListNode* reverseLinkLis(ListNode* head) {
  ListNode *dummy = new ListNode();
  ListNode *t = head; // 翻转后最后一个节点位置
  while(head && k--) {
    ListNode *p = head;
    head = head->next;
    p->next = dummy->next;
    dummy->next = p;
  }

  if (!head) {
    t->next = head;
  }

  head = dummy->next;
  delete dummy;
  return head;
}
```

3. 翻转链表的后k个节点

```
class Solution {
public:
  ListNode* rotateRight(ListNode* head, int k) {
    if (head == nullptr || k <= 0) {
      return head;
    }

    int       count = 1;
    ListNode* curr  = head;
    while (curr->next) {
      curr = curr->next;
      count++;
    }

    k = k % count;
    if (count <= 1 || k <= 0) {
      return head;
    }

    ListNode* dummy = new ListNode();
    dummy->next     = head;
    ListNode* prev  = dummy;
    ListNode* p     = head;
    for (int i = 1; i <= k && p; i++) {
      p = p->next;
    }

    ListNode* q = head;
    while (p) {
      prev = q;
      p    = p->next;
      q    = q->next;
    }

    prev->next = nullptr;
    head       = q;
    while (q->next) {
      q = q->next;
    }

    q->next = dummy->next;
    delete dummy;
    return head;
  }
};
```

4. 翻转链表的[m,n]区间内的节点 分析： 采用迭代算法:

5. 首先找到开始的第m-1个位置，即在哪个节点之后开始插入数据，记为prev；后续节点采用头插法的方式插入该节点之后。

6. 记`pstart = prev->next`为开始逆置的第一个节点，每次取`pstart`的下一个节点插入到`prev`之后。这样每次`pstart`的位置向后移动一个节点。

7. 对步骤2运行次数，为`n-m`，即需要逆置的字符串长度。

```
class Solution {
public:
  ListNode* reverseBetween(ListNode* head, int m, int n) {
    if (head == nullptr || m >= n) {
      return head;
    }

    if (m == n) return head;
    n -= m;
    ListNode prehead(0);
    prehead.next  = head;
    ListNode* pre = &prehead;
    while (--m) pre = pre->next;
    ListNode* pstart = pre->next;
    int       len    = n - m;
    while (len--) {
      ListNode* p  = pstart->next;
      pstart->next = p->next;
      p->next      = pre->next;
      pre->next    = p;
    }
    return prehead.next;
  }
};
```