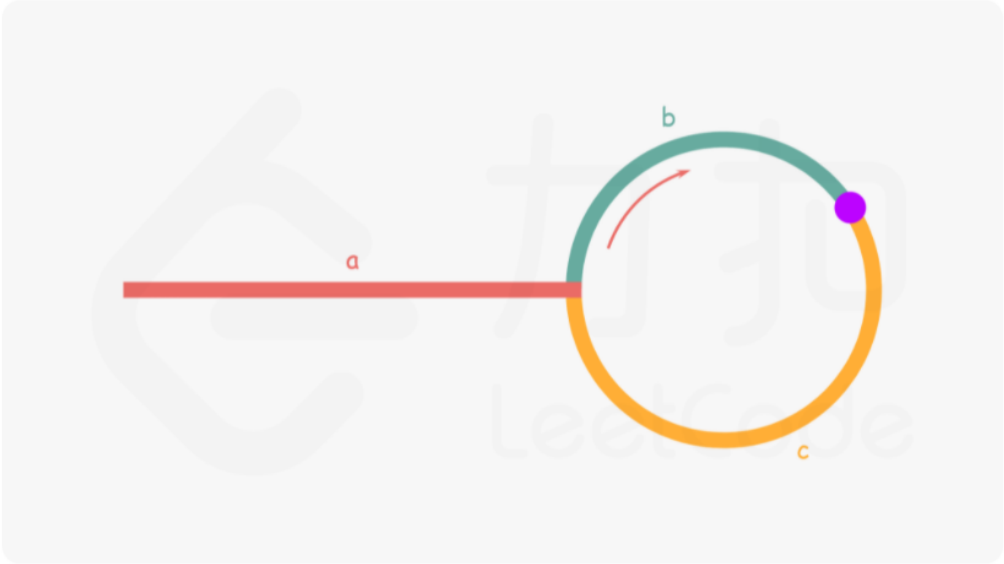
快慢指针的来源：(代码固定且简洁)

简介：

思路与算法

我们使用两个指针，$fast$ 与 $slow$。它们起始都位于链表的头部。随后，$slow$ 指针每次向后移动一个位置，而 $fast$ 指针向后移动两个位置。如果链表中存在环，则 $fast$ 指针最终将再次与 $slow$ 指针在环中相遇。

如下图所示，设链表中环外部分的长度为 a。$slow$ 指针进入环后，又走了 b 的距离与 $fast$ 相遇。此时，$fast$ 指针已经走完了环的 n 圈，因此它走过的总距离为 $a+n(b+c)+b=a+(n+1)b+nc$。



根据题意，任意时刻，$fast$ 指针走过的距离都为 $slow$ 指针的 2 倍。因此，我们有

$$a+(n+1)b+nc=2(a+b) \implies a=c+(n-1)(b+c)$$

有了 $a=c+(n-1)(b+c)$ 的等量关系，我们会发现：从相遇点到入环点的距离加上 $n-1$ 圈的环长，恰好等于从链表头部到入环点的距离。

因此，当发现 $slow$ 与 $fast$ 相遇时，我们再额外使用一个指针 ptr。起始，它指向链表头部；随后，它和 $slow$ 每次向后移动一个位置。最终，它们会在入环点相遇。

1. 慢指针一次走一步，快指针一次走两步
2. 若慢指针和快指针相遇则说明有环；若快慢指针不相遇则说明无环。
3. 快指针从相遇处出发，慢指针从起点出发，两者再次相遇的位置为入口处。

作用一：快慢指针找环

141. 环形链表

难度 简单 1194

给定一个链表，判断链表中是否有环。

如果链表中有某个节点，可以通过连续跟踪 `next` 指针再次到达，则链表中存在环。 为了表示给定链表中的环，我们使用整数 `pos` 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 `pos` 是 `-1`，则在该链表中没有环。**注意：** `pos` **不作为参数进行传递**，仅仅是为了标识链表的实际情况。

如果链表中存在环，则返回 `true` 。 否则，返回 `false` 。

进阶：

你能用 *O(1)*（即，常量）内存解决此问题吗？

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    bool hasCycle(ListNode *head) {
        //快慢指针
        ListNode*slow=head,*fast=head;
        if(!head)return false;
        do{
            slow=slow->next;
            fast=fast->next;
            if(fast)fast=fast->next;
        }while(fast&&slow!=fast);
        if(!fast)return false;
        return true;
    }
```

```
};
```

作用二:快慢指针找环的入口

142. 环形链表 II

难度 中等 1155

给定一个链表，返回链表开始入环的第一个节点。 如果链表无环，则返回 `null`。

为了表示给定链表中的环，我们使用整数 `pos` 来表示链表尾连接到链表中的位置（索引从 0 开始）。 如果 `pos` 是 `-1`，则在该链表中没有环。**注意，`pos` 仅仅是用于标识环的情况，并不会作为参数传递到函数中。**

说明：不允许修改给定的链表。

进阶：

- 你是否可以使用 `O(1)` 空间解决此题？

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        //快慢指针找入口
        ListNode*slow=head,*fast=head;
        if(!head)return nullptr;
        do{
            slow=slow->next;
            fast=fast->next;
            if(fast)fast=fast->next;
        }while(fast&&fast!=slow);
        if(!fast)return nullptr;
        slow=head;
        while(slow!=fast){
            slow=slow->next;
```

```
            fast=fast->next;
        }
        return slow;
    }
};
```