背景

Leetcode142题要求判断链表是否有环，并返回入环点。最先想到的是用set存取遍历过的节点，然后判断是否已遍历过，但这种解法会消耗额外的空间，网上看到一种不需要额外空间的解决方案，暂且叫快慢指针法吧，之前没接触过，所以把对这种方法的分析过程记录下来加深理解。

快慢指针法

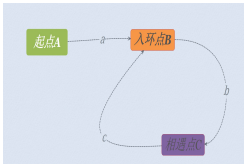
what

顾名思义，存两个指针slow,fast。两个指针都从表头开始走，slow 每次走一步，fast每次走两步，如果fast遇到null，则说明没有环，返回false；如果slow ==fast，说明有环，并且此时fast超了slow 整数n(n>=1)圈，返回true。

why，为什么快慢指针能相遇？

两个同学，同时从宿舍到操场跑圈，一个同学的速度是另一个同学的两倍，如果他们到操场后一直按自己的速度跑圈，速度快的同学就会实现套圈，也就是“相遇”。

how



fast的速度是slow 的两倍，所以时间相同的情况下，走过的距离也是两倍的关系；
当slow 指针到达相遇点C时，它走的距离为a+b;
此时fast指针已经在环中走了n圈（n>=1），假设一圈的距离为R（R=b+c），
则按下列步骤推导：

`2(a+b)=a+b+nR`

==> `a+b=n(b+c)`

==> `a=(n-1)(b+c)+c`

==> `a=(n-1)R+c`

最终得到 `a=(n-1)R+c` ,其中等式左边为slow 走的距离，等式右边为fast走的距离，他们的关系：slow 从A出发，fast从C出发，当slow 行走距离a到达入环点B时，fast也会到达入环点B，此时fast行走的距离为n-1圈加上c，其中n>=1。

注：网上很多博客都会推导出a=c的结论，即n=1,相遇时fast比slow 多走了1圈，其实这是不严谨的。比如slow 每次走1步，fast每次走2步，a为100步，b和c都为2步，此时slow 从head触发，fast从C出发，他们相遇时，fast在环里已经走了50圈，而不是1圈。

基于上面的分析，142题的快慢指针的解决方法也比较容易实现：

```
public class Solution {
    public ListNode detectCycle( ListNode head ) {
        if( head == null || head.next == null ){
         return null;
        }

        //1、寻找相遇点
        ListNode fp = head, sp = head;
        while( fp != null && fp.next != null){
            sp = sp.next;
            fp = fp.next.next;
            if( fp == sp ){
                break;
            }
        }

        //2、没有环的情况直接退出
        if( fp == null || fp.next == null ){
            return null;
        }

        //3、有环时，slow起点设置为head，fast从相遇点出发，相遇点即为入环点
        sp = head;
        while( fp != sp ){
            sp = sp.next;
            fp = fp.next;
        }
        return sp;
    }
}
```