链表+快慢指针：环形链表

题目来源：Leetcode 141：https://leetcode-cn.com/problems/linked-list-cycle/

暴力解法：二次到达

Java代码

链表实现代码：

```
class ListNode {
    int val;
    ListNode next;

    ListNode(int x) {
        val = x;
        next = null;
    }
}
```

```
/**
 * 解法一：二次到达解法
 * 1.定义数组记录已访问节点
 * new ListNode[10000];
 * 2.遍历链表的每个节点，并与容器中已存放的节点依次比较：
 * 相同则方法结束，返回true
 * 不同则存入最新位置，继续遍历下个节点
 * 3.若next指针为null，则方法结束，返回false
 *
 * @param head
 * @return
 */
public boolean hasCycle(ListNode head) {
    // 1.定义数组记录已访问节点
    ListNode[] array = new ListNode[10000];
    // 2.遍历链表的每个节点，
    while (head != null) {
        // 并与容器钟已存放的节点依次比较
        for (int i = 0; i < array.length; i++) {
            if (array[i] == head) {
                return true;
            }
            if (array[i] == null) {
                array[i] = head; // 将当前节点存放到最新位置
                break; // 结束容器的遍历
            }
        }

        head = head.next;
    }
    // 3.若next指针为null，则方法结束，返回false
```

```
        return false;
    }
```

最优解：追击问题（快慢指针）

java代码

```java
    /**
     * 解法二：快慢指针解法
     * 1.定义快慢两个指针：
     *  slow=head; fast=head.next;
     * 2.遍历链表：
     *  快指针步长为2：fast=fast.next.next;
     *  慢指针步长为1：slow=slow.next;
     * 3.当且仅当快慢指针重合，有环，返回true
     * 4.快指针为null，或其next指向null，没有环，返回false，操作结束
     * @param head
     * @return
     */
    public boolean hasCycle(ListNode head) {
        if (head == null) { // 链表中有节点[0, 10^4]个
            return false;
        }
        // 1.定义快慢两个指针：
        ListNode slow = head;
        ListNode fast = head.next;
        // 2.遍历链表：快指针步长为2，慢指针步长为1
        while (fast != null && fast.next != null) {
            // 3.当且仅当快慢指针重合：有环，操作结束
            if (slow == fast) {
                return true;
            }
            fast = fast.next.next; // 快指针步长为2
            slow = slow.next; // 慢指针步长为1
        } // 4.快指针为null，或其next指向null，没有环，返回false，操作结束
        return false;
    }
```

C++代码

```cpp
/**
 * 执行用时：8 ms, 在所有 C++ 提交中击败了 95.36% 的用户
 * 内存消耗：7.5 MB, 在所有 C++ 提交中击败了 54% 的用户
 */
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(NULL) {}
};

class Solution {
public:
```

```
    bool hasCycle(ListNode *head) {
        if (head == NULL) { // 链表中有节点[0, 10^4]个
            return false;
        }
        // 1.定义快慢两个指针：
        ListNode* slow = head;
        ListNode* fast = head->next;
        // 2.遍历链表：快指针步长为2，慢指针步长为1
        while (fast != NULL && fast->next != NULL) {
            // 3.当且仅当快慢指针重合：有环，操作结束
            if (slow == fast) {
                return true;
            }
            fast = fast->next->next; // 快指针步长为2
            slow = slow->next; // 慢指针步长为1
        } // 4.快指针为null，或其next指向null，没有环，返回false，操作结束
        return false;
    }
};
```

Python代码

```
# Definition for singly-linked list.
# class ListNode:
#     def __init__(self, x):
#         self.val = x
#         self.next = None

class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        if head is None:  # 链表中有节点[0, 10 ^ 4]个
            return False
        # 1.定义快慢两个指针：
        slow = head
        fast = head.next
        # 2.遍历链表：快指针步长为2，慢指针步长为1
        while fast is not None and fast.next is not None:
            # 3.当且仅当快慢指针重合：有环，操作结束
            if slow == fast:
                return True
            fast = fast.next.next  # 快指针步长为2
            slow = slow.next  # 慢指针步长为1
        # 4.快指针为null，或其next指向null，没有环，返回false，操作结束
        return False
```

测试用例

辅助数据结构：链表。代码如下：

```
    class ListNode {
        int val;
        ListNode next;

        ListNode(int x) {
            val = x;
            next = null;
        }
    }
```

输入：head = [1], pos = -1
输出：false
解释：pos 为环开始节点的索引，若pos = -1，则没有环。pos 不作为参数进行传递，仅仅是为了标识链表的实际情况

输入：head = [1,2], pos = 0
输出：true

输入：head = [3,2,0,-4], pos = 1
输出：true