```
                n = n1 + n2 + ca;
                pre = node;
                node = new Node(n % 10);
                node.next = pre;
                ca = n / 10;
                c1 = c1 != null ? c1.next : null;
                c2 = c2 != null ? c2.next : null;
        }
        if (ca == 1) {
                pre = node;
                node = new Node(1);
                node.next = pre;
        }
        reverseList(head1);
        reverseList(head2);
        return node;
}

public Node reverseList(Node head) {
        Node pre = null;
        Node next = null;
        while (head != null) {
                next = head.next;
                head.next = pre;
                pre = head;
                head = next;
        }
        return pre;
}
```

两个单链表相交的一系列问题

【题目】

在本题中，单链表可能有环，也可能无环。给定两个单链表的头节点 head1 和 head2，这两个链表可能相交，也可能不相交。请实现一个函数，如果两个链表相交，请返回相交的第一个节点；如果不相交，返回 null 即可。

要求：如果链表 1 的长度为 N，链表 2 的长度为 M，时间复杂度请达到 $O(N+M)$，额外空间复杂度请达到 $O(1)$。

【难度】

将 ★★★★

【解答】

这道题需要分析的情况非常多，同时因为有额外空间复杂度为 $O(1)$的限制，所以实现起来也比较困难。

本题可以拆分成三个子问题，每个问题都可以作为一道独立的算法题，具体如下。

问题一：如何判断一个链表是否有环，如果有，则返回第一个进入环的节点，没有则返回 null。

问题二：如何判断两个无环链表是否相交，相交则返回第一个相交节点，不相交则返回 null。

问题三：如何判断两个有环链表是否相交，相交则返回第一个相交节点，不相交则返回 null。

注意：如果一个链表有环，另外一个链表无环，它们是不可能相交的，直接返回 null。

下面逐一分析每个问题。

问题一：　如何判断一个链表是否有环，如果有，则返回第一个进入环的节点，没有则返回 null。

如果一个链表没有环，那么遍历链表一定可以遇到链表的终点；如果链表有环，那么遍历链表就永远在环里转下去了。如何找到第一个入环节点，具体过程如下：

1．设置一个慢指针 slow 和一个快指针 fast。在开始时，slow 和 fast 都指向链表的头节点 head。然后 slow 每次移动一步，fast 每次移动两步，在链表中遍历起来。

2．如果链表无环，那么 fast 指针在移动的过程中一定先遇到终点，一旦 fast 到达终点，说明链表是没有环的，直接返回 null，表示该链表无环，当然也没有第一个入环的节点。

3．如果链表有环，那么 fast 指针和 slow 指针一定会在环中的某个位置相遇，当 fast 和 slow 相遇时，fast 指针重新回到 head 的位置，slow 指针不动。接下来，fast 指针从每次移动两步改为每次移动一步，slow 指针依然每次移动一步，然后继续遍历。

4．fast 指针和 slow 指针一定会再次相遇，并且在第一个入环的节点处相遇。证明略。

注意：你也可以用哈希表完成问题一的判断，但是不符合题目关于空间复杂度的要求。

问题一的具体实现请参看如下代码中的 getLoopNode 方法。

```
public Node getLoopNode(Node head) {
        if (head == null || head.next == null || head.next.next == null) {
                return null;
        }
        Node n1 = head.next; // n1 -> slow
        Node n2 = head.next.next; // n2 -> fast
        while (n1 != n2) {
```

```
                if (n2.next == null || n2.next.next == null) {
                        return null;
                }
                n2 = n2.next.next;
                n1 = n1.next;
        }
        n2 = head; // n2 -> walk again from head
        while (n1 != n2) {
                n1 = n1.next;
                n2 = n2.next;
        }
        return n1;
}
```

如果解决了问题一，我们就知道了两个链表有环或者无环的情况。如果一个链表有环，另一个链表无环，那么这两个链表是无论如何也不可能相交的。能相交的情况就分为两种，一种是两个链表都无环，即问题二；另一种是两个链表都有环，即问题三。

问题二：如何判断两个无环链表是否相交，相交则返回第一个相交节点，不相交则返回 null。

如果两个无环链表相交，那么从相交节点开始，一直到两个链表终止的这一段，是两个链表共享的。解决问题二的具体过程如下：

1. 链表 1 从头节点开始，走到最后一个节点（不是结束），统计链表 1 的长度记为 len1，同时记录链表 1 的最后一个节点记为 end1。

2. 链表 2 从头节点开始，走到最后一个节点（不是结束），统计链表 2 的长度记为 len2，同时记录链表 2 的最后一个节点记为 end2。

3. 如果 end1!=end2，说明两个链表不相交，返回 null 即可；如果 end==end2，说明两个链表相交，进入步骤 4 来找寻第一个相交节点。

4. 如果链表 1 比较长，链表 1 就先走 len1−len2 步；如果链表 2 比较长，链表 2 就先走 len2−len1 步。然后两个链表一起走，一起走的过程中，两个链表第一次走到一起的那个节点，就是第一个相交的节点。

例如：链表 1 长度为 100，链表 2 长度为 30，如果已经由步骤 3 确定了链表 1 和链表 2 一定相交，那么接下来，链表 1 先走 70 步，然后链表 1 和链表 2 一起走，它们一定会共同进入第一个相交的节点。

问题二的具体实现请参看如下代码中的 noLoop 方法。

```
public Node noLoop(Node head1, Node head2) {
        if (head1 == null || head2 == null) {
                return null;
        }
```

```
        Node cur1 = head1;
        Node cur2 = head2;
        int n = 0;
        while (cur1.next != null) {
                n++;
                cur1 = cur1.next;
        }
        while (cur2.next != null) {
                n--;
                cur2 = cur2.next;
        }
        if (cur1 != cur2) {
                return null;
        }
        cur1 = n > 0 ? head1 : head2;
        cur2 = cur1 == head1 ? head2 : head1;
        n = Math.abs(n);
        while (n != 0) {
                n--;
                cur1 = cur1.next;
        }
        while (cur1 != cur2) {
                cur1 = cur1.next;
                cur2 = cur2.next;
        }
        return cur1;
}
```

问题三：如何判断两个有环链表是否相交，相交则返回第一个相交节点，不相交则返回 null。

考虑问题三的时候，我们已经得到了两个链表各自的第一个入环节点，假设链表 1 的第一个入环节点记为 loop1，链表 2 的第一个入环节点记为 loop2。以下是解决问题三的过程：

1．如果 loop1==loop2，那么两个链表的拓扑结构如图 2-4 所示。

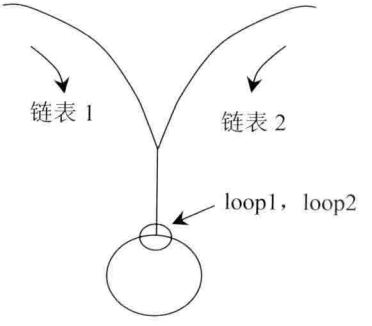

图 2-4

这种情况下，我们只要考虑链表 1 从头开始到 loop1 这一段与链表 2 从头开始到 loop2 这一段，在那里第一次相交即可，而不用考虑进环该怎么处理，这就与问题二类似，只不过问题二是把 null 作为一个链表的终点，而这里是把 loop1(loop2)作为链表的终点。但是判断的主要过程是相同的。

2．如果 loop1!=loop2，两个链表不相交的拓扑结构如图 2-5 所示。两个链表相交的拓扑结构如图 2-6 所示。

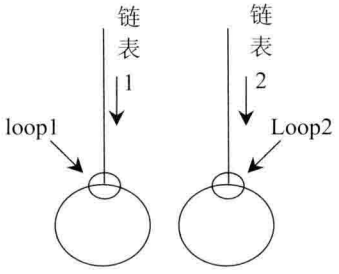

图 2-5

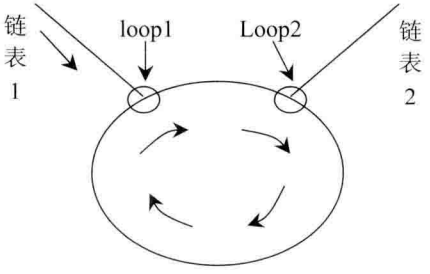

图 2-6

如何分辨是这两种拓扑结构的哪一种呢？进入步骤 3。

3．让链表 1 从 loop1 出发，因为 loop1 和之后的所有节点都在环上，所以将来一定能回到 loop1。如果回到 loop1 之前并没有遇到 loop2，说明两个链表的拓扑结构如图 2-5 所示，也就是不相交，直接返回 null；如果回到 loop1 之前遇到了 loop2，说明两个链表的拓扑结构如图 2-6 所示，也就是相交。因为 loop1 和 loop2 都在两条链表上，只不过 loop1 是离链表 1 较近的节点，loop2 是离链表 2 较近的节点。所以，此时返回 loop1 或 loop2 都可以。

问题三的具体实现参看如下代码中的 bothLoop 方法。

```
public Node bothLoop(Node head1, Node loop1, Node head2, Node loop2) {
        Node cur1 = null;
        Node cur2 = null;
        if (loop1 == loop2) {
                cur1 = head1;
                cur2 = head2;
                int n = 0;
                while (cur1 != loop1) {
                        n++;
                        cur1 = cur1.next;
                }
                while (cur2 != loop2) {
                        n--;
                        cur2 = cur2.next;
                }
                cur1 = n > 0 ? head1 : head2;
```

```
                cur2 = cur1 == head1 ? head2 : head1;
                n = Math.abs(n);
                while (n != 0) {
                        n--;
                        cur1 = cur1.next;
                }
                while (cur1 != cur2) {
                        cur1 = cur1.next;
                        cur2 = cur2.next;
                }
                return cur1;
        } else {
                cur1 = loop1.next;
                while (cur1 != loop1) {
                        if (cur1 == loop2) {
                                return loop1;
                        }
                        cur1 = cur1.next;
                }
                return null;
        }
}
```

全部过程参看如下代码中的 getIntersectNode 方法，这也是整个题目的主方法。

```
public class Node {
        public int value;
        public Node next;

        public Node(int data) {
                this.value = data;
        }
}

public Node getIntersectNode(Node head1, Node head2) {
        if (head1 == null || head2 == null) {
                return null;
        }
        Node loop1 = getLoopNode(head1);
        Node loop2 = getLoopNode(head2);
        if (loop1 == null && loop2 == null) {
                return noLoop(head1, head2);
        }
        if (loop1 != null && loop2 != null) {
                return bothLoop(head1, loop1, head2, loop2);
        }
        return null;
}
```