link: null
title: 珠峰架构师成长计划
description: null
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=55 sentences=87, words=576

1. 链表介绍

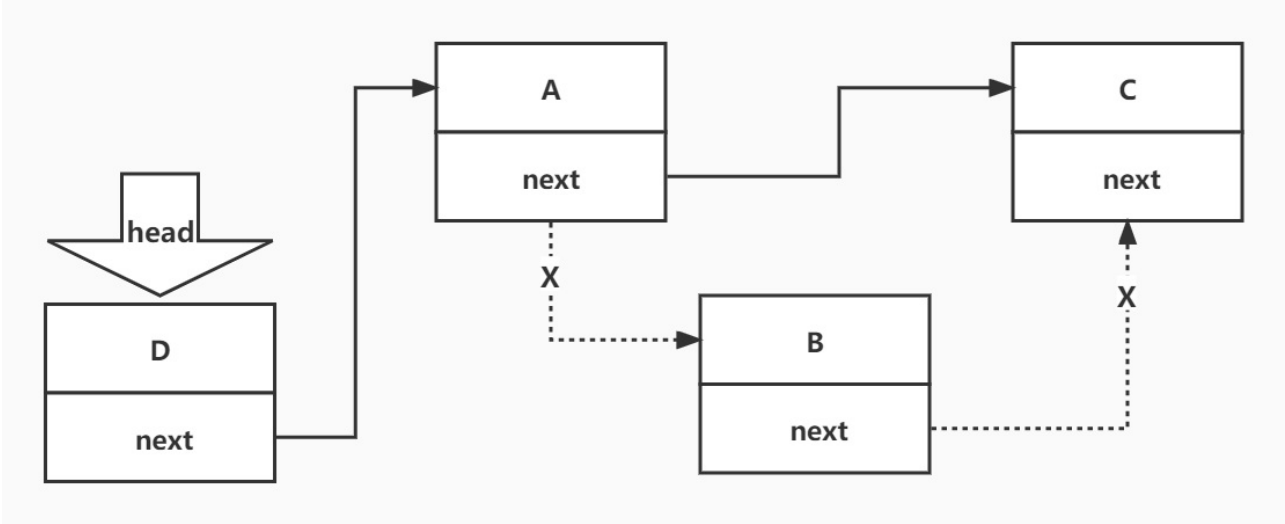
- 链表 (https://baike.baidu.com/item/%E9%93%BE%E8%A1%A8/9794473)是一种 **物理存储单元** 上 **非连续**、 **非顺序** 的存储结构
- 链表由一系列 **结点** (链表中每一个元素称为结点)组成
- 每个结点包括两个部分：一个是存储数据元素的 **数据域**，另一个是存储下一个结点地址的 **指针域**
- 数据元素的 **逻辑顺序** 是通过链表中的指针链接次序实现的

2.链表实现

2.1 地址实现

- diagraming (https://www.processon.com/diagraming/6189318d6376896480ef9baa)

```
class ListNode {
    constructor(data, next) {

        this.data = data;

        this.next = next;
    }
}

class List {
    constructor() {
        this.size = 0;
        this.head = null;
    }

    add(index, ListNode) {

        if (index === 0) {

            ListNode.next = this.head;

            this.head = ListNode;
        } else {

            let prev = this.get(index - 1);

            ListNode.next = prev.next;

            prev.next = ListNode;
        }

        this.size++;
    }

    rangeCheck(index) {
        if (index < 0 || index >= this.size) {
            throw new Error('索引越界');
        }
    }

    get(index) {
        this.rangeCheck(index);
        let curr = this.head;
        while (index--) {
            curr = curr.next;
        }
        return curr;
    }

    remove(index) {
        this.rangeCheck(index);
        if (index === 0) {
            this.head = this.head.next;
        } else {
            let prev = this.get(index - 1);
            prev.next = prev.next.next;
        }
    }

    clear() {
        this.head = null;
        this.size = 0;
    }

    print() {

        let curr = this.head;
        let str = '';
        while (curr) {

            str += curr.data + '->';

            curr = curr.next;
        }
        str += 'null';
        console.log(str);
    }
}

let list = new List();

let a = new ListNode('A');
list.add(0, a);

let c = new ListNode('C');
list.add(1, c);

let b = new ListNode('B');
list.add(1, b);

list.remove(1);

let d = new ListNode('D');
list.add(0, d);

list.print();
```



2.2 数组实现 #

- diagraming (https://www.processon.com/diagraming/61896d3c0791290c36870a83)

```
class List {
    constructor(head, value) {

        this.data = [];

        this.next = [];
        this.head = head;
        this.data[this.head] = value;
    }
    add(index, nextIndex, value) {
        this.next[index] = nextIndex;
        this.data[nextIndex] = value;
    }
    print() {

        let curr = this.head;
        let str = '';
        while (curr) {

            str += this.data[curr] + '->';

            curr = this.next[curr];
        }
        str += 'null';
        console.log(str);
    }
}
let head = 2;
let list = new List(head, 'A');
list.add(head, 4, 'B');
list.add(4, 6, 'C');
list.add(6, 0, 'D');
console.log(list.next.join(''));
console.log(list.data.join(''));
list.print();
```

3.leetcode

- linked-list-cycle (https://leetcode-cn.com/problems/linked-list-cycle/submissions/)
- processon (https://www.processon.com/diagraming/618952480791290c3686ffdf)

```
var hasCycle = function (head) {
    if (head === null) return false;
    let slow = head;
    let fast = head;
    while (fast.next && fast.next.next) {
        slow = slow.next;
        fast = fast.next.next;
        if (slow === fast)
            return true;
    }
    return false;
};
```

- linked-list-cycle-ii (https://leetcode-cn.com/problems/linked-list-cycle-ii)
- diagraming1 (https://www.processon.com/diagraming/61895a2c0e3e740b37456c25)
- diagraming2 (https://www.processon.com/diagraming/61896ba07d9c0828718ae912)

```
var detectCycle = function(head) {
    if (head === null) return head;
    let slow = head;
    let fast = head;
    let isCycle = false;
    while (fast.next && fast.next.next) {
        slow = slow.next;
        fast = fast.next.next;
        if (slow === fast){
            isCycle = true;
            break;
        }
    }
    if (!isCycle) {
        return null;
    }
    fast = head;
    while (slow !== fast) {
        slow = slow.next;
        fast = fast.next;
    }
    return slow;
};
```