自定义MyLinkedList\_ LeetCode\_707\_DesignLinkedList\_Easy\_Easy

# LeetCode\_707\_DesignLinkedList\_Easy\_Easy

## 题目介绍

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*LeetCode\_707\_DesignLinkedList\_Easy\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* LeetCode\_707\_DesignLinkedList\_Easy

\* https://leetcode.com/problems/design-linked-list/description/

\* 难度：Easy

\* DateTime：2018-10-007 13:22

\* 题目介绍：

\* Design your implementation of the linked list. You can choose to use the singly linked list or the doubly linked list. A node in a singly linked list should have two attributes: val and next. val is the value of the current node, and next is a pointer/reference to the next node. If you want to use the doubly linked list, you will need one more attribute prev to indicate the previous node in the linked list. Assume all nodes in the linked list are 0-indexed.

\*

\* Implement these functions in your linked list class:

\*

\* get(index) : Get the value of the index-th node in the linked list. If the index is invalid, return -1.

\* addAtHead(val) : Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list.

\* addAtTail(val) : Append a node of value val to the last element of the linked list.

\* addAtIndex(index, val) : Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted.

\* deleteAtIndex(index) : Delete the index-th node in the linked list, if the index is valid.

\* Example:

\*

\* MyLinkedList linkedList = new MyLinkedList();

\* linkedList.addAtHead(1);

\* linkedList.addAtTail(3);

\* linkedList.addAtIndex(1, 2); // linked list becomes 1->2->3

\* linkedList.get(1); // returns 2

\* linkedList.deleteAtIndex(1); // now the linked list is 1->3

\* linkedList.get(1); // returns 3

\* Note:

\*

\* All values will be in the range of [1, 1000].

\* The number of operations will be in the range of [1, 1000].

\* Please do not use the built-in LinkedList library.

## 思路分析

\* 思路分析：利用双向链表。定义一个头尾节点head和tail。

\* 都是一些基本的链表操作，很简单。

\* 自定义一个基本方法：

\* DoublyListNode getNodeAtIndex(int index)；返回index处的节点。

\* 其他很多方法都是利用该方法实现的。

## Java代码

class MyLinkedList {

DoublyListNode head;

DoublyListNode tail;

/\*\*

\* Initialize your data structure here.

\*/

public MyLinkedList() {

head = new DoublyListNode(0);

tail = new DoublyListNode(0);

head.next = tail;

tail.pre = head;

}

/\*\*

\* Get the value of the index-th node in the linked list. If the index is invalid, return -1.

\*/

public int get(int index) {

DoublyListNode node = getNodeAtIndex(index);

return (node == null || node == tail) ? -1 : node.val;

}

/\*\*

\* 自定义一个辅助方法：很重要

\*/

public DoublyListNode getNodeAtIndex(int index) {

if (index < 0) return null;

DoublyListNode temp = head;

for (int i = 0; i <= index; i++) {

temp = temp.next;

if (temp == null) return null;

}

return temp;//包括了len+1对应tail

}

/\*\*

\* Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list.

\*/

public void addAtHead(int val) {

DoublyListNode node = new DoublyListNode(val);

head.next.pre = node;

node.next = head.next;

head.next = node;

node.pre = head;

}

/\*\*

\* Append a node of value val to the last element of the linked list.

\*/

public void addAtTail(int val) {

DoublyListNode node = new DoublyListNode(val);

tail.pre.next = node;

node.pre = tail.pre;

node.next = tail;

tail.pre = node;

}

/\*\*

\* Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted.

\*/

public void addAtIndex(int index, int val) {

DoublyListNode node = getNodeAtIndex(index);

if (node == null) return;

if (node == tail) {

addAtTail(val);

return;

}

DoublyListNode newNode = new DoublyListNode(val);

node.pre.next = newNode;

newNode.pre = node.pre;

newNode.next = node;

node.pre = newNode;

}

/\*\*

\* Delete the index-th node in the linked list, if the index is valid.

\*/

public void deleteAtIndex(int index) {

DoublyListNode node = getNodeAtIndex(index);

if (node == null || node == tail) return;

node.pre.next = node.next;

node.next.pre = node.pre;

}

/\*\*

\* 共用的链表结构ListNode

\*/

private class DoublyListNode {

int val;

DoublyListNode pre;

DoublyListNode next;

DoublyListNode(int val) {

this.val = val;

}

}

/\*\*

\* 打印链表

\*/

public void printLinkedList() {

DoublyListNode headTemp = head.next;

StringBuilder sb = new StringBuilder();

while (headTemp != tail) {

sb.append(headTemp.val).append("-->");

headTemp = headTemp.next;

}

String r = sb.toString();

System.out.println(r.substring(0, sb.length() - 3));

}

}

/\*\*

\* Your MyLinkedList object will be instantiated and called as such:

\* MyLinkedList obj = new MyLinkedList();

\* int param\_1 = obj.get(index);

\* obj.addAtHead(val);

\* obj.addAtTail(val);

\* obj.addAtIndex(index,val);

\* obj.deleteAtIndex(index);

\*/

import org.junit.Test;

@Test

public void test() {

MyLinkedList linkedList = new MyLinkedList();

linkedList.addAtHead(1);

linkedList.printLinkedList();

linkedList.addAtTail(3);

linkedList.printLinkedList();

linkedList.addAtIndex(1, 2); // linked list becomes 1->2->3

linkedList.printLinkedList();

int val = linkedList.get(1); // returns 2

System.out.println("val = " + val);

linkedList.deleteAtIndex(1); // now the linked list is 1->3

linkedList.printLinkedList();

int val2 = linkedList.get(1); // returns 3

System.out.println("val2 = " + val2);

}