队列：最近的请求次数

题目来源：Leetcode 933 https://leetcode-cn.com/problems/number-of-recent-calls/

暴力解法：遍历数组

Java代码

```
    /**
     * 暴力解法
     * 1.创建数组，存放所有的请求
     *   整型数组，存放10000个元素
     * 2.把当前请求存入数组
     *   记录最后一次存入的索引，从0开始
     * 3.统计距离此次请求前3000毫秒之间的请求次数
     *   从最后一次存放位置倒序遍历
     *
     * @param t 时长为 t（单位：毫秒） 的请求
     * @return 过去3000毫秒内有多少次请求：[t-3000, t]
     */
    public int ping(int t) {
        // 2.把当前请求存入数组
        int end = 0;
        for (int i = 0; i < array.length; i++) {
            if (array[i] == 0) { // 细节：数组元素是0，该位置没有存过请
求
                array[i] = t;
                end = i; // 记录最近一次请求存放的索引
                break;
            }
        }
        // 3.统计距离此次请求前3000毫秒之间的请求次数
        int count = 0; // 计数器
        while (array[end] >= t - 3000) {
            count++;
            if (--end < 0) { // 防止越界
                break;
            }
        }
```

```
        return count;
    }
```

优化解法：计算起止索引差值

java代码

```
    // 1.创建数组，存放所有的请求
    // int[] array = new int[10000];
    int[] array = new int[3002];
    // 2.记录起止索引，从0开始
    int start = 0, end = 0;
    /**
     * 优化解法：双指针
     * 1.创建数组存放请求：int[3002]
     * 2.额外定义开始指针
     *  start=0，end=0，记录起止索引
     * 3.存放请求后，更新起止索引
     *  end++; 从上次的开始索引（start）向后查找
     *  直到新的合法的起始位置
     * 4.通过end与start差值计算请求次数
     *
     * @param t 时长为 t（单位：毫秒） 的请求
     * @return 过去3000毫秒内有多少次请求：[t-3000, t]
     */
    public int ping(int t) {
        // 3.存放请求后，更新起止索引
        array[end++] = t; // 存放最近一次请求，结束索引加 1
        end = end == array.length ? 0 : end; // 越界后，从0开始
        // 从start位置开始，正向查找符合要求的请求次数
        while (array[start] < t - 3000) { // 过滤掉所有不符合要求的数
据
            start ++;
            start = start == array.length ? 0 : start;
        }
        // 4.通过end与start差值计算请求次数
        if (start > end) { // 请求次数超过数组容量，发生了溢出
            return array.length - (start - end);
        }
        // 此时，end为最新一次请求 + 1 的索引，start是3000毫秒前的第一次合
法请求索引
        return end - start;
```

```
    }
```

最优解：队列解法

java代码

```
    /**
     * 最优解：队列解法
     * 1.使用链表实现一个队列
     *   定义属性：队头-head、队尾-tail、长度-size
     *   定义方法：添加节点-add(int)、移除节点-poll() 、队列长度-size()
     *   定义内部类：Node，封装每次入队的请求数据和指向下一个节点的指针
     * 2.每次请求向队列尾部追加节点
     * 3.循环检查队头数据是否合法
     *   不合法则移除该节点
     * 4.返回队列长度
     * @param t
     * @return
     */
    public int ping(int t) {
        // 2.每次请求向队列尾部追加节点
        q.add(t);
        // 3.循环检查队头数据是否合法
        while (q.head.getVal() < t - 3000)
            q.poll();
        // 4.返回队列长度
        return q.size();
    }
```

队列实现代码：

```
    Queue q;
    public RecentCounter() {
        q = new Queue();
    }

    class Queue { // 1.使用链表实现一个队列
        Node head;
        Node tail;
        int size = 0;

        public Queue() {
```

```
        }

        public void add(int x) { // 向尾部添加一个节点
            Node last = tail;
            Node newNode = new Node(x);
            tail = newNode; // 尾指针指向新节点
            if (last == null) { // 第一次添加数据
                head = newNode;
                tail = newNode;
            } else {
                last.next = newNode; // 前一个节点指向新节点
            }
            size++; // 每添加一个节点，队列长度+1
        }

        public int poll() { // 从头部移除一个节点
            int headVal = head.val; // 获取头节点的数据
            Node next = head.next;
            head.next = null; // 链表第一个节点断开
            head = next; // head指针指向后一个节点
            if (next == null) { // 队列中的最后一个元素
                tail = null;  // 处理尾指针
            }
            size--; // 每移出一个节点，队列长度减1
            return headVal;
        }

        public int size() {
            return size;
        }

        class Node { // 队列节点：链表结构
            int val;
            Node next;
            Node(int x) {
                val = x;
            }
            int getVal() {
                return val;
            }
        }
    }
```

C++代码

最优解代码：

```
class RecentCounter {
public:
    RecentCounter() {

    }

    int ping(int t) {
        q.push(t);
        while(q.front() < t-3000) q.pop();
        return q.size();
    }
private:
    queue<int> q;
};

/**
 * Your RecentCounter object will be instantiated and called as
such:
 * RecentCounter* obj = new RecentCounter();
 * int param_1 = obj->ping(t);
 */
```

Python代码

队列实现代码：

```
class Node: # 队列节点：链表结构
    def __init__(self, x: int):
        self.val = x
        self.next = None

    def getVal(self):
        return self.val
class Queue :
    def __init__(self):
        self.head=None
        self.tail=None
        self.size=0
    def add(self,x :int): # 向尾部添加一个节点
```

```
        last=self.tail
        newNode= Node(x)
        self.tail=newNode # 尾指针指向新节点
        if not last: # 第一次添加数据
            self.head=newNode
            self.tail=newNode
        else:
            last.next=newNode # 前一个节点指向新节点
        self.size+=1 # 每添加一个节点，队列长度+1
    def poll(self):
        headVal = self.head.val # 获取头节点的数据
        next=self.head.next
        self.head=None # 链表第一个节点断开
        self.head=next # head指针指向后一个节点
        if not next: # 队列中的最后一个元素
            self.tail=None # 处理尾指针
        self.size-=1 # 每移出一个节点，队列长度减1
        return headVal
    def size(self):
        return self.size
```

最优解代码：

```
class RecentCounter:
    def __init__(self):
        self.q=Queue()

    def ping(self, t: int) -> int:
        self.q.add(t)
        while self.q.head.getVal()<t-3000:
            self.q.poll()
        return self.q.size
```

测试用例

辅助数据结构：队列。代码如下：

```
class Queue { // 1.使用链表实现一个队列
    Node head;
    Node tail;
    int size = 0;

    public Queue() {
```

```
    }

    public void add(int x) {}
    public int poll() {}
    public int size() {}

    class Node { // 队列节点：链表结构
        int val;
        Node next;
        Node(int x) {
            val = x;
        }
        int getVal() {
            return val;
        }
    }
}
```

```
输入：inputs = [[1],[100],[3001],[3002]]
输出：[1,2,3,3]
```