# Leetcode 题解 - 栈和队列

* [Leetcode 题解 - 栈和队列](#leetcode-题解---栈和队列)
  + [1. 用栈实现队列](#1-用栈实现队列)
  + [2. 用队列实现栈](#2-用队列实现栈)
  + [3. 最小值栈](#3-最小值栈)
  + [4. 用栈实现括号匹配](#4-用栈实现括号匹配)
  + [5. 数组中元素与下一个比它大的元素之间的距离](#5-数组中元素与下一个比它大的元素之间的距离)
  + [6. 循环数组中比当前元素大的下一个元素](#6-循环数组中比当前元素大的下一个元素)

## 1. 用栈实现队列

232. Implement Queue using Stacks (Easy)

[Leetcode](https://leetcode.com/problems/implement-queue-using-stacks/description/) / [力扣](https://leetcode-cn.com/problems/implement-queue-using-stacks/description/)

栈的顺序为后进先出，而队列的顺序为先进先出。使用两个栈实现队列，一个元素需要经过两个栈才能出队列，在经过第一个栈时元素顺序被反转，经过第二个栈时再次被反转，此时就是先进先出顺序。

class MyQueue {  
  
 private Stack<Integer> in = new Stack<>();  
 private Stack<Integer> out = new Stack<>();  
  
 public void push(int x) {  
 in.push(x);  
 }  
  
 public int pop() {  
 in2out();  
 return out.pop();  
 }  
  
 public int peek() {  
 in2out();  
 return out.peek();  
 }  
  
 private void in2out() {  
 if (out.isEmpty()) {  
 while (!in.isEmpty()) {  
 out.push(in.pop());  
 }  
 }  
 }  
  
 public boolean empty() {  
 return in.isEmpty() && out.isEmpty();  
 }  
}

## 2. 用队列实现栈

225. Implement Stack using Queues (Easy)

[Leetcode](https://leetcode.com/problems/implement-stack-using-queues/description/) / [力扣](https://leetcode-cn.com/problems/implement-stack-using-queues/description/)

在将一个元素 x 插入队列时，为了维护原来的后进先出顺序，需要让 x 插入队列首部。而队列的默认插入顺序是队列尾部，因此在将 x 插入队列尾部之后，需要让除了 x 之外的所有元素出队列，再入队列。

class MyStack {  
  
 private Queue<Integer> queue;  
  
 public MyStack() {  
 queue = new LinkedList<>();  
 }  
  
 public void push(int x) {  
 queue.add(x);  
 int cnt = queue.size();  
 while (cnt-- > 1) {  
 queue.add(queue.poll());  
 }  
 }  
  
 public int pop() {  
 return queue.remove();  
 }  
  
 public int top() {  
 return queue.peek();  
 }  
  
 public boolean empty() {  
 return queue.isEmpty();  
 }  
}

## 3. 最小值栈

155. Min Stack (Easy)

[Leetcode](https://leetcode.com/problems/min-stack/description/) / [力扣](https://leetcode-cn.com/problems/min-stack/description/)

class MinStack {  
  
 private Stack<Integer> dataStack;  
 private Stack<Integer> minStack;  
 private int min;  
  
 public MinStack() {  
 dataStack = new Stack<>();  
 minStack = new Stack<>();  
 min = Integer.MAX\_VALUE;  
 }  
  
 public void push(int x) {  
 dataStack.add(x);  
 min = Math.min(min, x);  
 minStack.add(min);  
 }  
  
 public void pop() {  
 dataStack.pop();  
 minStack.pop();  
 min = minStack.isEmpty() ? Integer.MAX\_VALUE : minStack.peek();  
 }  
  
 public int top() {  
 return dataStack.peek();  
 }  
  
 public int getMin() {  
 return minStack.peek();  
 }  
}

对于实现最小值队列问题，可以先将队列使用栈来实现，然后就将问题转换为最小值栈，这个问题出现在 编程之美：3.7。

## 4. 用栈实现括号匹配

20. Valid Parentheses (Easy)

[Leetcode](https://leetcode.com/problems/valid-parentheses/description/) / [力扣](https://leetcode-cn.com/problems/valid-parentheses/description/)

"()[]{}"  
  
Output : true

public boolean isValid(String s) {  
 Stack<Character> stack = new Stack<>();  
 for (char c : s.toCharArray()) {  
 if (c == '(' || c == '{' || c == '[') {  
 stack.push(c);  
 } else {  
 if (stack.isEmpty()) {  
 return false;  
 }  
 char cStack = stack.pop();  
 boolean b1 = c == ')' && cStack != '(';  
 boolean b2 = c == ']' && cStack != '[';  
 boolean b3 = c == '}' && cStack != '{';  
 if (b1 || b2 || b3) {  
 return false;  
 }  
 }  
 }  
 return stack.isEmpty();  
}

## 5. 数组中元素与下一个比它大的元素之间的距离

739. Daily Temperatures (Medium)

[Leetcode](https://leetcode.com/problems/daily-temperatures/description/) / [力扣](https://leetcode-cn.com/problems/daily-temperatures/description/)

Input: [73, 74, 75, 71, 69, 72, 76, 73]  
Output: [1, 1, 4, 2, 1, 1, 0, 0]

在遍历数组时用栈把数组中的数存起来，如果当前遍历的数比栈顶元素来的大，说明栈顶元素的下一个比它大的数就是当前元素。

public int[] dailyTemperatures(int[] temperatures) {  
 int n = temperatures.length;  
 int[] dist = new int[n];  
 Stack<Integer> indexs = new Stack<>();  
 for (int curIndex = 0; curIndex < n; curIndex++) {  
 while (!indexs.isEmpty() && temperatures[curIndex] > temperatures[indexs.peek()]) {  
 int preIndex = indexs.pop();  
 dist[preIndex] = curIndex - preIndex;  
 }  
 indexs.add(curIndex);  
 }  
 return dist;  
}

## 6. 循环数组中比当前元素大的下一个元素

503. Next Greater Element II (Medium)

[Leetcode](https://leetcode.com/problems/next-greater-element-ii/description/) / [力扣](https://leetcode-cn.com/problems/next-greater-element-ii/description/)

Input: [1,2,1]  
Output: [2,-1,2]  
Explanation: The first 1's next greater number is 2;  
The number 2 can't find next greater number;  
The second 1's next greater number needs to search circularly, which is also 2.

与 739. Daily Temperatures (Medium) 不同的是，数组是循环数组，并且最后要求的不是距离而是下一个元素。

public int[] nextGreaterElements(int[] nums) {  
 int n = nums.length;  
 int[] next = new int[n];  
 Arrays.fill(next, -1);  
 Stack<Integer> pre = new Stack<>();  
 for (int i = 0; i < n \* 2; i++) {  
 int num = nums[i % n];  
 while (!pre.isEmpty() && nums[pre.peek()] < num) {  
 next[pre.pop()] = num;  
 }  
 if (i < n){  
 pre.push(i);  
 }  
 }  
 return next;  
}