

宫水三叶的刷题日记

# 表达式计算

Author : 宫水三叶

Date : 2021/10/07

QQ Group: 703311589

WeChat : oaoaya

宫水三叶

刷题日记

公众号: 宫水三叶的刷题日记

噔噔噔噔，这是公众号「[宫水三叶的刷题日记](#)」的原创专题「表达式计算问题」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「表达式计算问题」即可获取最新下载链接。

💡下面介绍使用本合集的最佳使用实践：

## 学习算法：

1. 打开在线目录（[Github 版](#) & [Gitee 版](#)）；
2. 从侧边栏的类别目录找到「表达式计算问题」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题；
4. 拿到题号之后，回到本合集进行检索。

## 维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流   

## 题目描述

这是 LeetCode 上的 [150. 逆波兰表达式求值](#)，难度为 中等。

Tag：「表达式计算」

根据 逆波兰表示法，求表达式的值。

有效的算符包括  $+$ 、 $-$ 、 $*$ 、 $/$ 。每个运算对象可以是整数，也可以是另一个逆波兰表达式。

说明：

- 整数除法只保留整数部分。
- 给定逆波兰表达式总是有效的。换句话说，表达式总会得出有效数值且不存在除数为 0 的情况。

### 示例 1：

输入：tokens = ["2","1","+","3","\*"]

输出：9

解释：该算式转化为常见的中缀算术表达式为： $((2 + 1) * 3) = 9$

### 示例 2：

输入：tokens = ["4","13","5","/","+"]

输出：6

解释：该算式转化为常见的中缀算术表达式为： $(4 + (13 / 5)) = 6$

### 示例 3：

输入：tokens = ["10","6","9","3","+","-11","\*","/","\*","17","+","5","+"]

输出：22

解释：

该算式转化为常见的中缀算术表达式为：

$((10 * (6 / ((9 + 3) * -11))) + 17) + 5$   
 $= ((10 * (6 / (12 * -11))) + 17) + 5$   
 $= ((10 * (6 / -132)) + 17) + 5$   
 $= ((10 * 0) + 17) + 5$   
 $= (0 + 17) + 5$   
 $= 17 + 5$   
 $= 22$

### 提示：

- $1 \leq \text{tokens.length} \leq 10^4$
- tokens[i] 要么是一个算符（"+"、 "-"、 "\*" 或 "/"），要么是一个在范围 [-200, 200] 内的整数

### 逆波兰表达式：

逆波兰表达式是一种后缀表达式，所谓后缀就是指算符写在后面。

- 平常使用的算式则是一种中缀表达式，如  $(1 + 2) * (3 + 4)$ 。
- 该算式的逆波兰表达式写法为  $((1 2 +) (3 4 +) *)$ 。

逆波兰表达式主要有以下两个优点：

刷题日记

公众号：宫水三叶的刷题日记

- 去掉括号后表达式无歧义，上式即便写成  $1\ 2 + 3\ 4 + *$  也可以依据次序计算出正确结果。
- 适合用栈操作运算：遇到数字则入栈；遇到算符则取出栈顶两个数字进行计算，并将结果压入栈中。

## 基本思路

这是一道关于「表达式计算」的题目。

所有的「表达式计算」问题都离不开「栈」。

对于本题，我们可以建立一个「数字栈」，存放所有的数字，当遇到运算符时，从栈中取出两个数进行运算，并将结果放回栈内，整个过程结束后，栈顶元素就是最终结果。

而栈的实现通常有两种：使用数组继续模拟 & 使用系统自带的栈结构

## 数组模拟栈解法

执行结果： **通过** [显示详情 >](#)

执行用时： **3 ms** ，在所有 Java 提交中击败了 **99.81%** 的用户

内存消耗： **37.9 MB** ，在所有 Java 提交中击败了 **94.55%** 的用户

炫耀一下：



写题解，分享我的解题思路

代码：

刷题日记

公众号：宫水三叶的刷题日记

```
class Solution {
    public int evalRPN(String[] ts) {
        int[] d = new int[ts.length];
        int hh = 0, tt = -1;
        for (String s : ts) {
            if ("+-*/".contains(s)) {
                int b = d[tt--], a = d[tt--];
                d[++tt] = calc(a, b, s);
            } else {
                d[++tt] = Integer.parseInt(s);
            }
        }
        return d[tt];
    }
    int calc(int a, int b, String op) {
        if (op.equals("+")) return a + b;
        else if (op.equals("-")) return a - b;
        else if (op.equals("*")) return a * b;
        else if (op.equals("/")) return a / b;
        else return -1;
    }
}
```

- 时间复杂度： $O(n)$
- 空间复杂度： $O(n)$

宫水三叶  
の  
刷题日记

公众号: 宫水三叶的刷题日记

## 自带栈解法

执行结果： **通过** [显示详情 >](#)

执行用时： **6 ms** ，在所有 Java 提交中击败了 **89.27%** 的用户

内存消耗： **38.2 MB** ，在所有 Java 提交中击败了 **56.00%** 的用户

炫耀一下：



[✍ 写题解，分享我的解题思路](#)

代码：

```
class Solution {
    public int evalRPN(String[] ts) {
        Deque<Integer> d = new ArrayDeque<>();
        for (String s : ts) {
            if ("+-*/".contains(s)) {
                int b = d.pollLast(), a = d.pollLast();
                d.addLast(calc(a, b, s));
            } else {
                d.addLast(Integer.parseInt(s));
            }
        }
        return d.pollLast();
    }
    int calc(int a, int b, String op) {
        if (op.equals("+")) return a + b;
        else if (op.equals("-")) return a - b;
        else if (op.equals("*")) return a * b;
        else if (op.equals("/")) return a / b;
        else return -1;
    }
}
```

- 时间复杂度： $O(n)$
- 空间复杂度： $O(n)$

宫水三叶  
刷题日记

公众号：宫水三叶的刷题日记

## 其他

关于「表达式计算」，类似的题目在上周的「每日一题」也出现过：

- [224. 基本计算器](#)：包含符号 `+ - ( )`
- [227. 基本计算器 II](#)：包含符号 `+ - * /`
- [772. 基本计算器 III](#)：有锁题，包含符号 `+ - * / ( )`
- [770. 基本计算器 IV](#)：包含自定义函数符号

更多精彩内容，欢迎关注：[公众号](#) / [Github](#) / [LeetCode](#) / [知乎](#)

## 题目描述

这是 LeetCode 上的 [224. 基本计算器](#)，难度为 中等。

Tag：「表达式计算」

给你一个字符串表达式 `s`，请你实现一个基本计算器来计算并返回它的值。

示例 1：

输入：`s = "1 + 1"`  
输出：2

示例 2：

输入：`s = "2-1 + 2 "`  
输出：3

示例 3：

输入：`s = "(1+(4+5+2)-3)+(6+8)"`  
输出：23

提示：

宫水三叶  
刷题日记

公众号：宫水三叶的刷题日记

- $1 \leq s.length \leq 3 \times 10^5$
- $s$  由数字、`'+'`、`'-'`、`'('`、`')'`、和 `' '` 组成
- $s$  表示一个有效的表达式

---

## 双栈解法

我们可以使用两个栈 `nums` 和 `ops`。

- `nums`：存放所有的数字
- `ops`：存放所有的数字以外的操作，`+/-` 也看做是一种操作

然后从前往后做，对遍历到的字符做分情况讨论：

- 空格：跳过
- `(`：直接加入 `ops` 中，等待与之匹配的 `)`
- `)`：使用现有的 `nums` 和 `ops` 进行计算，直到遇到左边最近的一个左括号为止，计算结果放到 `nums`
- 数字：从当前位置开始继续往后取，将整个连续数字整体取出，加入 `nums`
- `+/-`：需要将操作放入 `ops` 中。在放入之前先把栈内可以算的都算掉，使用现有的 `nums` 和 `ops` 进行计算，直到没有操作或者遇到左括号，计算结果放到 `nums`

一些细节：

- 由于第一个数可能是负数，为了减少边界判断。一个小技巧是先往 `nums` 添加一个 `0`
- 为防止 `()` 内出现的首个字符为运算符，将所有的空格去掉，并将 `(-` 替换为 `(0-`，`(+` 替换为 `(0+`（当然也可以不进行这样的预处理，将这个处理逻辑放到循环里去做）

代码：

宫水三叶  
の  
刷题日记

公众号：宫水三叶的刷题日记



```

class Solution {
    public int calculate(String s) {
        // 存放所有的数字
        Deque<Integer> nums = new ArrayDeque<>();
        // 为了防止第一个数为负数，先往 nums 加个 0
        nums.addLast(0);
        // 将所有空格去掉
        s = s.replaceAll(" ", "");
        // 存放所有的操作，包括 +/−
        Deque<Character> ops = new ArrayDeque<>();
        int n = s.length();
        char[] cs = s.toCharArray();
        for (int i = 0; i < n; i++) {
            char c = cs[i];
            if (c == '(') {
                ops.addLast(c);
            } else if (c == ')') {
                // 计算到最近一个左括号为止
                while (!ops.isEmpty()) {
                    char op = ops.peekLast();
                    if (op != '(') {
                        calc(nums, ops);
                    } else {
                        ops.pollLast();
                        break;
                    }
                }
            } else {
                if (isNum(c)) {
                    int u = 0;
                    int j = i;
                    // 将从 i 位置开始后面的连续数字整体取出，加入 nums
                    while (j < n && isNum(cs[j])) u = u * 10 + (int)(cs[j++] - '0');
                    nums.addLast(u);
                    i = j - 1;
                } else {
                    if (i > 0 && (cs[i - 1] == '(' || cs[i - 1] == '+' || cs[i - 1] == '-')) {
                        nums.addLast(0);
                    }
                    // 有一个新操作要入栈时，先把栈内可以算的都算了
                    while (!ops.isEmpty() && ops.peekLast() != '(') calc(nums, ops);
                    ops.addLast(c);
                }
            }
        }
        while (!ops.isEmpty()) calc(nums, ops);
    }
}

```

```

        return nums.peekLast();
    }
    void calc(Deque<Integer> nums, Deque<Character> ops) {
        if (nums.isEmpty() || nums.size() < 2) return;
        if (ops.isEmpty()) return;
        int b = nums.pollLast(), a = nums.pollLast();
        char op = ops.pollLast();
        nums.addLast(op == '+' ? a + b : a - b);
    }
    boolean isNum(char c) {
        return Character.isDigit(c);
    }
}

```

- 时间复杂度： $O(n)$
- 空间复杂度： $O(n)$

## 进阶

1. 如果在此基础上，再考虑 `*` 和 `/`，需要增加什么考虑？如何维护运算符的优先级？
2. 在 1 的基础上，如果考虑支持自定义符号，例如 `a / func(a, b) * (c + d)`，需要做出什么调整？

## 补充

1. 对应进阶 1 的补充。

一个支持 `+ - * / ^ %` 的「计算器」，基本逻辑是一样的，使用字典维护一个符号优先级：

宫水三叶  
の  
刷题日记

公众号: 宫水三叶的刷题日记

```

class Solution {
    Map<Character, Integer> map = new HashMap<>(){
        put('-', 1);
        put('+', 1);
        put('*', 2);
        put('/', 2);
        put('%', 2);
        put('^', 3);
    };
    public int calculate(String s) {
        s = s.replaceAll(" ", "");
        char[] cs = s.toCharArray();
        int n = s.length();
        Deque<Integer> nums = new ArrayDeque<>();
        nums.addLast(0);
        Deque<Character> ops = new ArrayDeque<>();
        for (int i = 0; i < n; i++) {
            char c = cs[i];
            if (c == '(') {
                ops.addLast(c);
            } else if (c == ')') {
                while (!ops.isEmpty()) {
                    if (ops.peekLast() != '(') {
                        calc(nums, ops);
                    } else {
                        ops.pollLast();
                        break;
                    }
                }
            } else {
                if (isNumber(c)) {
                    int u = 0;
                    int j = i;
                    while (j < n && isNumber(cs[j])) u = u * 10 + (cs[j++] - '0');
                    nums.addLast(u);
                    i = j - 1;
                } else {
                    if (i > 0 && (cs[i - 1] == '(' || cs[i - 1] == '+' || cs[i - 1] == '-' || cs[i - 1] == '^')) {
                        nums.addLast(0);
                    }
                    while (!ops.isEmpty() && ops.peekLast() != '(') {
                        char prev = ops.peekLast();
                        if (map.get(prev) >= map.get(c)) {
                            calc(nums, ops);
                        } else {
                            break;
                        }
                    }
                }
            }
        }
        return nums.pollLast();
    }
}

```

```

        }
    }
    ops.addLast(c);
}
}
}
while (!ops.isEmpty() && ops.peekLast() != '(') calc(nums, ops);
return nums.peekLast();
}

void calc(Deque<Integer> nums, Deque<Character> ops) {
    if (nums.isEmpty() || nums.size() < 2) return;
    if (ops.isEmpty()) return;
    int b = nums.pollLast(), a = nums.pollLast();
    char op = ops.pollLast();
    int ans = 0;
    if (op == '+') {
        ans = a + b;
    } else if (op == '-') {
        ans = a - b;
    } else if (op == '*') {
        ans = a * b;
    } else if (op == '/') {
        ans = a / b;
    } else if (op == '^') {
        ans = (int) Math.pow(a, b);
    } else if (op == '%') {
        ans = a % b;
    }
    nums.addLast(ans);
}

boolean isNumber(char c) {
    return Character.isDigit(c);
}
}

```

2. 关于进阶 2，其实和进阶 1 一样，重点在于维护优先级。但还有一些编码细节：

对于非单个字符的运算符（例如 函数名 function），可以在处理前先将所有非单字符的运算符进行替换（将 function 替换为 @# 等）

然后对特殊运算符做特判，确保遍历过程中识别到特殊运算符之后，往后整体读入（如 function(a,b) -> @(a, b)，@(a, b) 作为整体处理）

刷题日记

公众号: 宫水三叶的刷题日记

## 题目描述

这是 LeetCode 上的 [227. 基本计算器 II](#)，难度为 **中等**。

Tag：「表达式计算」

给你一个字符串表达式  $s$ ，请你实现一个基本计算器来计算并返回它的值。

整数除法仅保留整数部分。

示例 1：

```
输入：s = "3+2*2"  
输出：7
```

示例 2：

```
输入：s = " 3/2 "  
输出：1
```

示例 3：

```
输入：s = " 3+5 / 2 "  
输出：5
```

提示：

- $1 \leq s.length \leq 3 \times 10^5$
- $s$  由整数和算符 ('+', '-', '\*', '/') 组成，中间由一些空格隔开
- $s$  表示一个有效表达式
- 表达式中的所有整数都是非负整数，且在范围  $[0, 2^{31} - 1]$  内
- 题目数据保证答案是一个 32-bit 整数

# 双栈解法

如果你有看这篇 [题解](#) 的话，今天这道题就是道练习题。

帮你巩固 双栈解决「通用表达式」问题的通用解法。

事实上，我提供这套解决方案不仅仅能解决只有  $+ - ( )$  ([224. 基本计算器](#)) 或者  $+ - * /$  ([227. 基本计算器 II](#)) 的表达式问题，还能解决  $+ - * / ^ \% ( )$  的完全表达式问题。

甚至支持自定义运算符，只要在运算优先级上进行维护即可。

对于「表达式计算」这一类问题，你都可以使用这套思路进行解决。我十分建议你加强理解这套处理逻辑。

对于「任何表达式」而言，我们都使用两个栈 `nums` 和 `ops`：

- `nums`：存放所有的数字
- `ops`：存放所有的数字以外的操作

然后从前往后做，对遍历到的字符做分情况讨论：

- 空格：跳过
- `(`：直接加入 `ops` 中，等待与之匹配的 `)`
- `)`：使用现有的 `nums` 和 `ops` 进行计算，直到遇到左边最近的一个左括号为止，计算结果放到 `nums`
- 数字：从当前位置开始继续往后取，将整个连续数字整体取出，加入 `nums`
- $+ - * / ^ \%$ ：需要将操作放入 `ops` 中。在放入之前先把栈内可以算的都算掉（只有「栈内运算符」比「当前运算符」优先级高/同等，才进行运算），使用现有的 `nums` 和 `ops` 进行计算，直到没有操作或者遇到左括号，计算结果放到 `nums`

我们可以通过 🍊 来理解 只有「栈内运算符」比「当前运算符」优先级高/同等，才进行运算 是什么意思：

因为我们是从前往后做的，假设我们当前已经扫描到  $2 + 1$  了（此时栈内的操作为  $+$ ）。

1. 如果后面出现的  $+ 2$  或者  $- 1$  的话，满足「栈内运算符」比「当前运算符」优先级高/同等，可以将  $2 + 1$  算掉，把结果放到 `nums` 中；
2. 如果后面出现的是  $* 2$  或者  $/ 1$  的话，不满足「栈内运算符」比「当前运算符」

优先级高/同等，这时候不能计算  $2 + 1$ 。

一些细节：

- 由于第一个数可能是负数，为了减少边界判断。一个小技巧是先往 `nums` 添加一个 0
- 为防止 () 内出现的首个字符为运算符，将所有的空格去掉，并将 `(-` 替换为 `(0-`，`(+` 替换为 `(0+`（当然也可以不进行这样的预处理，将这个处理逻辑放到循环里去做）
- 从理论上分析，`nums` 最好存放的是 `long`，而不是 `int`。因为可能存在 `大数 + 大数 + 大数 + ... - 大数 - 大数` 的表达式导致中间结果溢出，最终答案不溢出的情况

代码：

宫水三叶  
の  
刷题日记

公众号：宫水三叶的刷题日记

```

class Solution {
    // 使用 map 维护一个运算符优先级
    // 这里的优先级划分按照「数学」进行划分即可
    Map<Character, Integer> map = new HashMap<>(){
        put('-', 1);
        put('+', 1);
        put('*', 2);
        put('/', 2);
        put('%', 2);
        put('^', 3);
    };
    public int calculate(String s) {
        // 将所有空格去掉
        s = s.replaceAll(" ", "");
        char[] cs = s.toCharArray();
        int n = s.length();
        // 存放所有的数字
        Deque<Integer> nums = new ArrayDeque<>();
        // 为了防止第一个数为负数，先往 nums 加个 0
        nums.addLast(0);
        // 存放所有「非数字以外」的操作
        Deque<Character> ops = new ArrayDeque<>();
        for (int i = 0; i < n; i++) {
            char c = cs[i];
            if (c == '(') {
                ops.addLast(c);
            } else if (c == ')') {
                // 计算到最近一个左括号为止
                while (!ops.isEmpty()) {
                    if (ops.peekLast() != '(') {
                        calc(nums, ops);
                    } else {
                        ops.pollLast();
                        break;
                    }
                }
            } else {
                if (isNumber(c)) {
                    int u = 0;
                    int j = i;
                    // 将从 i 位置开始后面的连续数字整体取出，加入 nums
                    while (j < n && isNumber(cs[j])) u = u * 10 + (cs[j++] - '0');
                    nums.addLast(u);
                    i = j - 1;
                } else {
                    if (i > 0 && (cs[i - 1] == '(' || cs[i - 1] == '+' || cs[i - 1] == '-' || cs[i - 1] == '*' || cs[i - 1] == '/')) {
                        calc(nums, ops);
                    }
                    ops.addLast(c);
                }
            }
        }
        return nums.pollLast();
    }
}

```



```

        nums.addLast(0);
    }
    // 有一个新操作要入栈时，先把栈内可以算的都算了
    // 只有满足「栈内运算符」比「当前运算符」优先级高/同等，才进行运算
    while (!ops.isEmpty() && ops.peekLast() != '(') {
        char prev = ops.peekLast();
        if (map.get(prev) >= map.get(c)) {
            calc(nums, ops);
        } else {
            break;
        }
    }
    ops.addLast(c);
}
}

// 将剩余的计算完
while (!ops.isEmpty()) calc(nums, ops);
return nums.peekLast();
}

void calc(Deque<Integer> nums, Deque<Character> ops) {
    if (nums.isEmpty() || nums.size() < 2) return;
    if (ops.isEmpty()) return;
    int b = nums.pollLast(), a = nums.pollLast();
    char op = ops.pollLast();
    int ans = 0;
    if (op == '+') ans = a + b;
    else if (op == '-') ans = a - b;
    else if (op == '*') ans = a * b;
    else if (op == '/') ans = a / b;
    else if (op == '^') ans = (int)Math.pow(a, b);
    else if (op == '%') ans = a % b;
    nums.addLast(ans);
}

boolean isNumber(char c) {
    return Character.isDigit(c);
}
}

```

- 时间复杂度： $O(n)$
- 空间复杂度： $O(n)$

宫水三叶  
の  
刷题日记

公众号：宫水三叶的刷题日记

## 总结

还记得我在 [题解](#) 留的「进阶」内容？

1. 如果 `+` `-` 基础上，再考虑 `*` 和 `/`，需要增加什么考虑？如何维护运算符的优先级？

这个进阶问题就对应了 LeetCode 上的两道题：

- [227. 基本计算器 II](#)：本题，包含符号 `+` `-` `*` `/`
- [772. 基本计算器 III](#)：有锁题，包含符号 `+` `-` `*` `/` `(` `)`

2. 在「问题1」的基础上，如果考虑支持自定义符号，例如 `a / func(a, b) * (c + d)`，需要做出什么调整？

这个进阶问题，在 LeetCode 上也有类似的题目：

- [770. 基本计算器 IV](#)：包含自定义函数符号

综上，使用三叶提供的这套「双栈通用解决方案」，可以解决所有的「表达式计算」问题。因为这套「表达式计算」处理逻辑，本质上模拟了人脑的处理逻辑：根据下一位的运算符优先级决定当前运算符是否可以马上计算。

---

\*\*🔍 更多精彩内容，欢迎关注：[公众号](#) / [Github](#) / [LeetCode](#) / [知乎](#) \*\*

## 题目描述

这是 LeetCode 上的 [1006. 笨阶乘](#)，难度为 **中等**。

Tag：「数学」、「栈」

通常，正整数  $n$  的阶乘是所有小于或等于  $n$  的正整数的乘积。

例如， $\text{factorial}(10) = 10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 * 2 * 1$ 。

相反，我们设计了一个笨阶乘 `clumsy`：在整数的递减序列中，我们以一个固定顺序的操作符序列来依次替换原有的乘法操作符：乘法(`*`)，除法(`/`)，加法(`+`)和减法(`-`)。

例如， $\text{clumsy}(10) = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1$ 。然而，这些运算仍然使用通常的算术运

算顺序：我们在任何加、减步骤之前执行所有的乘法和除法步骤，并且按从左到右处理乘法和除法步骤。

另外，我们使用的除法是地板除法（floor division），所以  $10 * 9 / 8$  等于 11。这保证结果是一个整数。

实现上面定义的笨函数：给定一个整数 N，它返回 N 的笨阶乘。

示例 1：

输入：4

输出：7

解释： $7 = 4 * 3 / 2 + 1$

示例 2：

输入：10

输出：12

解释： $12 = 10 * 9 / 8 + 7 - 6 * 5 / 4 + 3 - 2 * 1$

提示：

- $1 \leq N \leq 10000$
- $-2^{31} \leq \text{answer} \leq 2^{31} - 1$  （答案保证符合 32 位整数）

## 通用表达式解法

第一种解法是我们的老朋友解法了，使用「双栈」来解决通用表达式问题。

事实上，我提供这套解决方案不仅仅能解决只有  $+$   $-$   $( )$ （[224. 基本计算器](#)）或者  $+$   $-$   $*$   $/$ （[227. 基本计算器 II](#)）的表达式问题，还能解决  $+$   $-$   $*$   $/$   $^$   $\%$   $( )$  的完全表达式问题。

甚至支持自定义运算符，只要在运算优先级上进行维护即可。

对于「表达式计算」这一类问题，你都可以使用这套思路进行解决。我十分建议你加强理解这套处理逻辑。

对于「任何表达式」而言，我们都使用两个栈 `nums` 和 `ops`：

- `nums`：存放所有的数字
- `ops`：存放所有的数字以外的操作

然后从前往后做，对遍历到的字符做分情况讨论：

- 空格：跳过
- `(`：直接加入 `ops` 中，等待与之匹配的 `)`
- `)`：使用现有的 `nums` 和 `ops` 进行计算，直到遇到左边最近的一个左括号为止，计算结果放到 `nums`
- 数字：从当前位置开始继续往后取，将整个连续数字整体取出，加入 `nums`
- `+ - * / ^ %`：需要将操作放入 `ops` 中。在放入之前先把栈内可以算的都算掉（只有「栈内运算符」比「当前运算符」优先级高/同等，才进行运算），使用现有的 `nums` 和 `ops` 进行计算，直到没有操作或者遇到左括号，计算结果放到 `nums`

我们可以通过 🍎 来理解 只有「栈内运算符」比「当前运算符」优先级高/同等，才进行运算 是什么意思：

因为我们是从前往后做的，假设我们当前已经扫描到 `2 + 1` 了（此时栈内的操作为 `+`）。

1. 如果后面出现的 `+ 2` 或者 `- 1` 的话，满足「栈内运算符」比「当前运算符」优先级高/同等，可以将 `2 + 1` 算掉，把结果放到 `nums` 中；
2. 如果后面出现的是 `* 2` 或者 `/ 1` 的话，不满足「栈内运算符」比「当前运算符」优先级高/同等，这时候不能计算 `2 + 1`。

更为详细的讲解可以看这篇题解：[使用「双栈」解决「究极表达式计算」问题](#)

代码：

宫水三叶  
の  
刷题日记

公众号：宫水三叶的刷题日记

```

class Solution {
    public int clumsy(int n) {
        Deque<Integer> nums = new ArrayDeque<>();
        Deque<Character> ops = new ArrayDeque<>();
        // 维护运算符优先级
        Map<Character, Integer> map = new HashMap<>(){
            put('*', 2);
            put('/', 2);
            put('+', 1);
            put('-', 1);
        };
        char[] cs = new char[]{'*', '/', '+', '-'};
        for (int i = n, j = 0; i > 0; i--, j++) {
            char op = cs[j % 4];
            nums.addLast(i);
            // 如果「当前运算符优先级」不高于「栈顶运算符优先级」，说明栈内的可以算
            while (!ops.isEmpty() && map.get(ops.peekLast()) >= map.get(op)) {
                calc(nums, ops);
            }
            if (i != 1) ops.add(op);
        }
        // 如果栈内还有元素没有算完，继续算
        while (!ops.isEmpty()) calc(nums, ops);
        return nums.peekLast();
    }

    void calc(Deque<Integer> nums, Deque<Character> ops) {
        int b = nums.pollLast(), a = nums.pollLast();
        int op = ops.pollLast();
        int ans = 0;
        if (op == '+') ans = a + b;
        else if (op == '-') ans = a - b;
        else if (op == '*') ans = a * b;
        else if (op == '/') ans = a / b;
        nums.addLast(ans);
    }
}

```

- 时间复杂度： $O(n)$
- 空间复杂度： $O(n)$

宫水三叶

刷题日记

公众号: 宫水三叶的刷题日记

## 数学解法（打表技巧分析）

这次在讲【证明】之前，顺便给大家讲讲找规律的题目该怎么做。

由于是按照特定顺序替换运算符，因此应该是有一些特性可以被我们利用的。

通常我们需要先实现一个可打表的算法（例如上述的解法一，这是为什么掌握「通用表达式」解法具有重要意义），将连续数字的答案打印输出，来寻找规律：

```
Solution solution = new Solution();
for (int i = 1; i <= 10000; i++) {
    int res = solution.clumsy(i);
    System.out.println(i + " : " + res);
}
```

宫水三叶  
の  
刷题日记

公众号：宫水三叶的刷题日记

✓ Tests passed: 1 of 1 test – 2 s 160 ms

400 : 401  
401 : 403  
402 : 404  
403 : 402  
404 : 405  
405 : 407  
406 : 408  
407 : 406  
408 : 409  
409 : 411  
410 : 412  
411 : 410  
412 : 413  
413 : 415  
414 : 416  
415 : 414  
416 : 417  
417 : 419  
418 : 420  
419 : 418  
420 : 421  
421 : 423  
422 : 424  
423 : 422  
424 : 425  
425 : 427  
426 : 428  
427 : 426  
428 : 429  
429 : 431  
430 : 432  
431 : 430  
432 : 433  
433 : 435  
434 : 436  
435 : 434  
436 : 437  
437 : 439  
438 : 440  
439 : 438

似乎  $n$  与 答案比较接近，我们考虑将两者的差值输出：

刷题日记

公众号: 宫水三叶的刷题日记

```
Solution solution = new Solution();
for (int i = 1; i <= 10000; i++) {
    int res = solution.clumsy(i);
    System.out.println(i + " : " + res + " : " + (res - i));
}
```

✓ Tests passed: 1 of 1 test - 2 s 272 ms

```
400 : 401 : 1
401 : 403 : 2
402 : 404 : 2
403 : 402 : -1
404 : 405 : 1
405 : 407 : 2
406 : 408 : 2
407 : 406 : -1
408 : 409 : 1
409 : 411 : 2
410 : 412 : 2
411 : 410 : -1
412 : 413 : 1
413 : 415 : 2
414 : 416 : 2
415 : 414 : -1
416 : 417 : 1
417 : 419 : 2
418 : 420 : 2
419 : 418 : -1
420 : 421 : 1
421 : 423 : 2
422 : 424 : 2
423 : 422 : -1
424 : 425 : 1
425 : 427 : 2
426 : 428 : 2
427 : 426 : -1
428 : 429 : 1
429 : 431 : 2
430 : 432 : 2
431 : 430 : -1
432 : 433 : 1
433 : 435 : 2
434 : 436 : 2
435 : 434 : -1
436 : 437 : 1
437 : 439 : 2
438 : 440 : 2
439 : 438 : -1
```

咦，好像发现了什么不得了的东西。似乎每四个数，差值都是 [1, 2, 2, -1]

再修改我们的打表逻辑，来验证一下（只输出与我们猜想不一样的数字）：



```

Solution solution = new Solution();
int[] diff = new int[]{1,2,2,-1};
for (int i = 1; i <= 10000; i++) {
    int res = solution.clumsy(i);
    int t = res - i;
    if (t != diff[i % 4]) {
        System.out.println(i + " : " + res);
    }
}

```

```

1 : 1
2 : 2
3 : 6
4 : 7

```

只有前四个数字被输出，其他数字都是符合我们的猜想规律的。

到这里我们已经知道代码怎么写可以 AC 了，十分简单。

代码：

```

class Solution {
    public int clumsy(int n) {
        int[] special = new int[]{1,2,6,7};
        int[] diff = new int[]{1,2,2,-1};
        if (n <= 4) return special[(n - 1) % 4];
        return n + diff[n % 4];
    }
}

```

- 时间复杂度： $O(1)$
- 空间复杂度： $O(1)$

## 证明

讲完我们的【实战技巧】之后，再讲讲如何证明。

上述的做法比较适合于笔试或者比赛，但是面试，通常还需要证明做法为什么是正确的。

我们不失一般性的分析某个  $n$ ，当然这个  $n$  必须是大于 4，不属于我们的特判值。

然后对  $n$  进行讨论（根据我们的打表猜想去证明规律是否可推广）：

1.  $n \% 4 == 0$  :  $f(n) = n * (n - 1) / (n - 2) + \dots + 5 - 4 * 3 / 2 + 1 = n + 1$ ，即  $\text{diff} = 1$
2.  $n \% 4 == 1$  :  $f(n) = n * (n - 1) / (n - 2) + \dots + 6 - 5 * 4 / 3 + 2 - 1 = n + 2$ ，即  $\text{diff} = 2$
3.  $n \% 4 == 2$  :  $f(n) = n * (n - 1) / (n - 2) + \dots + 7 - 6 * 5 / 4 + 3 - 2 * 1 = n + 2$ ，即  $\text{diff} = 2$
4.  $n \% 4 == 3$  :  $f(n) = n * (n - 1) / (n - 2) + \dots + 8 - 7 * 6 / 5 + 4 - 3 * 2 / 1 = n - 1$ ，即  $\text{diff} = -1$

上述的表达式展开过程属于小学数学内容，省略号部分的项式的和为 0，因此你只需要关注我写出来的那部分。

至此，我们证明了我们的打表猜想具有「可推广」的特性。

甚至我们应该学到：证明可以是基于猜想去证明，而不必从零开始进行推导。

---

更多精彩内容，欢迎关注：[公众号](#) / [Github](#) / [LeetCode](#) / [知乎](#)

💡更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。

最新专题合集资料下载，可关注公众号「[宫水三叶的刷题日记](#)」，后台回复「表达式计算问题」获取下载链接。

觉得专题不错，可以请作者吃糖 🍬🍬🍬：

宫水三叶  
の  
刷题日记

公众号: 宫水三叶的刷题日记



“给作者手机充个电”

YOLO 的赞赏码

版权声明：任何形式的转载请保留出处 [Wiki](#)。