题1：[基本计算器](https://leetcode.cn/problems/basic-calculator/description/?utm_source=LCUS&utm_medium=ip_redirect&utm_campaign=transfer2china)

s由数字、'+'、'-'、'('、')'、和空格组成，求解其值

解法1：遇到括号就递归

class Solution

{

public:

    int i = 0;

    string ss;

    int calculate(string s)

    {

        ss = "";

        for (char x : s)

            if (x != ' ')

                ss += x;

        return dfs();

    }

    int dfs()

    {

        int ans = 0;

        while (i < ss.size())

        {

            int flag = 1;

            if (ss[i] == '-')

                flag = -1, i++;

            else if (ss[i] == '+')

                flag = 1, i++;

            int t;

            if (ss[i] == '(')

            {

                i++;

                t = dfs();

            }

            else if (ss[i] == ')')

            {

                i++;

                return ans;

            }

            else

            {

                t = ss[i] - '0';

                i++;

                while (i < ss.size() && ss[i] >= '0' && ss[i] <= '9')

                {

                    t \*= 10;

                    t += ss[i] - '0';

                    i++;

                }

            }

            t \*= flag;

            ans += t;

        }

        return ans;

    }

};

解法2：双栈

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)

{

    // 将所有的空格去掉

    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] == '-'))

                {

                    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);

}

}

作者：宫水三叶

链接：[双栈](https://leetcode.cn/problems/basic-calculator-ii/solutions/648832/shi-yong-shuang-zhan-jie-jue-jiu-ji-biao-c65k/?utm_source=LCUS&utm_medium=ip_redirect&utm_campaign=transfer2china)

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，将整一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 + ）。

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；

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

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。