1. 题目
2. 题干

实现一个基本的计算器来计算一个简单的字符串表达式的值。

字符串表达式仅包含非负整数，+， - ，\*，/ 四种运算符和空格  。 整数除法仅保留整数部分。

1. 示例

示例 1:

输入: "3+2\*2"

输出: 7

示例 2:

输入: " 3/2 "

输出: 1

示例 3:

输入: " 3+5 / 2 "

输出: 5

说明：

你可以假设所给定的表达式都是有效的。

请不要使用内置的库函数 eval。

1. 题解
2. 思路

通过栈的思路解决计算器问题，之前我们解决了一个没有括号的计算器问题，那么不会牵扯到递归的问题，当需要考虑括号时，就需要通过递归来解决！当遇到左括号时，就会进入一个新的递归中，当遇到右括号时，就会回溯到上一个递归中。其他的思路与基本的计算器题解是一样的。这里同样使用一个变量来存储上一个加减乘除运算符的值。使用一个全局变量i来存储索引。

递归思路：

判断当前的字符是否是空格，如果是，则跳过，进入下一个循环；

接着判断当前的字符是否是数字，如果是，则要考虑数字有多个位数的情况，构建数字；

判断当前的字符是否是左括号，如果是，则增加索引，并且进入下一个递归中，同时还要考虑当下一个递归回溯时，该左括号对应的右括号已经被判断了，因此需要再次递增索引；

判断当前的索引是否是右括号，如果是，则递归回溯，break；

判断当前的字符是否是加减乘除或者是最后一个数字，这个判断是啥意思呢？因为一个多位数字的结束一定是通过一个加减乘除符号或者是最后一位数字来标识的，因此，这个判断表示的就是数字已经构建完毕，现在可以根据上一个运算符来进行入栈出栈操作了。每次进行完一次入栈出栈操作后，都需要将num变量设为0，以构建下一个多位数字。同时将operation设为当前的运算符，以应对下一次运算符操作。注意，另一种情况是该字符是最后一位数字，那么我们就需要回溯，break。

在每个递归的最后，对栈中的数字进行累加并返回！！！！

1. 代码实现

Java：

class Solution {

    int i = 0;

    public int calculate(String s) {

        return dfs(s.trim());

    }

    public int dfs(String s){

        Stack<Integer> stack = new Stack<Integer>();

        int num =  0;

        char ch = '+';

        for(i = 0; i < s.length(); ++i){

            if(s.charAt(i) == ' '){

                continue;

            }

            if(Character.isDigit(s.charAt(i))){

                num = num \* 10 + s.charAt(i)-'0';

            }

            else if( s.charAt(i) == '('){

                i++;

                dfs(s);

                i++;

            }else if( s.charAt(i) == ')'){

                break;

            }

            if(!Character.isDigit(s.charAt(i)) || i== s.length()-1){

                if(ch == '+'){

                    stack.push(num);

                }

                else if( ch == '-'){

                    stack.push(-num);

                }

                else if(ch == '\*'){

                    stack.push(stack.pop() \* num);

                }

                else if(ch == '/'){

                    stack.push(stack.pop() / num);

                }

                //到达最后一位数字

                if(i == s.length()-1){

                    break;

                }

                ch = s.charAt(i);

                num = 0;

            }

        }

        int sum = 0;

        while(!stack.isEmpty()){

            sum += stack.pop();

        }

        return sum;

    }

}