中缀表达式/后缀表达式

运算符在数字中间用称为中缀表达式

作用：实现计算机功能

要点1:将字符串转换成数字和符号列表（可以转int，可以转String）

要点2:符号优先级相同时也要出栈操作

要点3:从后往前出栈操作即可

```
public class MidCalculator {

    public static void main(String[] args) {
        String str = "70+2*6-2";
        ArrStack numStack = new ArrStack(20);
        ArrStack signStack = new ArrStack(10);

        int curIndex = 0;
        String keepNum = "";
        char curChar ;

        while(true){

            curChar = str.substring(curIndex, curIndex + 1).charAt(0);
            //判断数值还是操作符
            if(isOper(curChar)){
                //栈为空的情况下直接加
                if (signStack.isEmpty()) {
                    signStack.push(curChar);
                }else {
                //优先级后面的小于等于前面则出栈操作否则直接入栈
                if(priority(curChar) <= priority(signStack.peek())){
                    int signPop = signStack.pop();
                    int next = numStack.pop();
                    int pre = numStack.pop();
                    int result = calculate(next, pre, signPop);
                    numStack.push(result);
                    signStack.push(curChar);
                }else{
                    signStack.push(curChar);
                }
                }
            }else{
                keepNum += curChar;
                //若是最后一个直接入栈
                if(curIndex == str.length()-1){
                    numStack.push(Integer.parseInt(keepNum));
                }else {
                //下一个若非操作符则继续加数字长度
                    char next = str.substring(curIndex + 1, curIndex + 2).charAt(0);
                    if (isOper(next)) {
                        numStack.push(Integer.parseInt(keepNum));
                        keepNum ="";
```

```
                    }
                }
            }
            curIndex ++;
            if(curIndex == str.length())break;
        }
        while(true){
            if(signStack.isEmpty())break;
            int sign = signStack.pop();
            int next = numStack.pop();
            int pre = numStack.pop();
            int calculate = calculate(next, pre, sign);
            numStack.push(calculate);
        }
        System.out.println(numStack.arr[0]);


    }
    public static boolean isOper(int val){
        return val == '+' || val == '-' || val == '*' || val == '/';
    }
    //返回运算符的优先级，优先级是程序员来确定，优先级使用数字表示
    //数字越大，则优先级越高
    public static int priority(int oper) {
        if (oper == '*' || oper == '/') {
            return 1;
        } else if (oper == '+' || oper == '-') {
            return 0;
        } else {
            return -1;//假定目前的表达式只有+，-，*，/
        }
    }

    public static int calculate(int next,int pre,int oper){
        int result = 0;
        switch (oper){
            case '+':
                result = pre+next;
                break;
            case '-':
                result = pre-next;
                break;
            case '*':
                result = pre*+next;
                break;
            case '/':
                result = pre/next;
                break;
            default:
                break;
        }
        return result;
    }
}

class ArrStack{
    int maxSize;
    int [] arr;
    int top =-1;

    public ArrStack(int maxSize){
        this.maxSize=maxSize;
        arr = new int [maxSize];
    }

    public boolean isEmpty(){
        return top == -1;
    }

    public boolean isFull(){
        return maxSize == top +1;
    }

    public void push(int num){
        arr[++top] = num;
    }
```

```
    public int pop(){
        int value = arr[top];
        top--;
        return value;
    }

    //显示栈的情况【遍历栈】,遍历时需要从栈顶开始显示
    public void list() {
        if (isEmpty()) {
            System.out.println("栈空，没有数据");
            return;
        }
        for (int i = top; i >= 0; i--) {
            System.out.printf("stack[%d]=%d\n", i, arr[i]);
        }
    }
    public int peek(){
        return arr[top];
    }
}
```

符号在数字后面称为后缀表达式：

解决有括号的计算

要点1:先以列表的形式转成中缀表达式 ==> 转后缀表达 ==> 计算

后缀表达式没有括号思路：

将中缀表达式“1+((2+3)×4)-5”转换为后缀表达式的过程如下 因此结果为：“123+4×+5-”

扫描到的元素	s2(栈底->栈顶)	s1(栈底->栈顶)	说明
1	1	空	数字，直接入栈
+	1	+	s1为空，运算符直接入栈
(	1	+(	左括号，直接入栈
(	1	+((	同上
2	1 2	+((	数字
+	1 2	+((+	s1栈顶为左括号，运算符直接入栈
3	1 2 3	+((+	数字
)	1 2 3 +	+(	右括号，弹出运算符直至遇到左括号
×	1 2 3 +	+(×	s1栈顶为左括号，运算符直接入栈
4	1 2 3 + 4	+(×	数字
)	1 2 3 + 4 ×	+	右括号，弹出运算符直至遇到左括号
-	1 2 3 + 4 × +	-	-与+优先级相同，因此弹出+，再压入-
5	1 2 3 + 4 × + 5	-	数字
到达最右端	1 2 3 + 4 × + 5 -	空	s1中剩余的运算符

计算思路如下：

（3+4）×5-6对应的后缀表达式就是34+5×6-，针对后缀表达式求值步骤如下：

从左至右扫描，将3和4压入堆栈； 遇到+运算符，因此弹出4和3（4为栈顶元素，3为次顶元素），计算出3+4的值，得7，再将7入栈； 将5入栈； 接下来是×运算符，因此弹出5和7，计算出7×5=35，将35入栈； 将6入栈； 最后是．运算符，计算出35-6的值，即29，由此得出最终结果

```
public class BackExpresstion {

    public static void main(String[] args) {
        String expression = "1+((2+3)*4)-5";
        //中缀表达式
        ArrayList<String> mid = getMidExpress(expression);
        System.out.println("mid = " + mid);
        //后缀表达式
```

```
        ArrayList<String> back = getBackExpress(mid);
        System.out.println("back = " + back);
        //计算结果
        int result = calculate(back);
        System.out.println("result = " + result);
    }

    private static int calculate(ArrayList<String> back) {

        Stack<String> stack = new Stack<>();
        int front = 0;
        int last = 0;
        int result = 0;
        for (String str : back) {
            //数字进stack
            if(str.matches("\\d")){
                stack.add(str);
            }else{
            // 符号操作
                last = Integer.parseInt(stack.pop());
                front = Integer.parseInt(stack.pop());
                if(str.equals("+")){
                    result  = last + front;
                }else if(str.equals("-")){
                    result  = front - last;
                }else if(str.equals("*")){
                    result  = last * front;
                }else if(str.equals("/")){
                    result  = front / last;
                }
                stack.add(result+"");
            }

        }



        return Integer.parseInt(stack.pop());
    }

    private static ArrayList<String> getBackExpress(ArrayList<String> mid) {
        //两个栈，s1最终，s2放符号
        ArrayList<String> s1 = new ArrayList<>(20);
        Stack<String> s2 = new Stack<>();
        for (String str:mid) {
            //对栈进行操作
            //数值直接放s1
            if(str.matches("\\d+")){
                s1.add(str);
            }else {
            //操作符若s2为空或栈顶为做括号直接进入s2
                if(s2.isEmpty() || s2.peek().equals("(")){
                    s2.add(str);
                }else {
                //若是右括号，弹出s2到左括号
                    if(str.equals(")")){
                        String tmp = "";
                        while (!"(".equals(tmp=s2.pop())){
                            s1.add(tmp);
                        }
                    }else {
                    //否则判断与s2栈顶相对应的优先级，若高于s2栈顶的符号则直接入栈，否则弹出s2栈顶符号到s1，此str入栈
                        if(priority(str) > priority(s2.peek())){
                            s2.add(str);
                        }else {
                            s1.add(s2.pop());
                            s2.add(str);
                        }
                    }


                }

            }

        }
```

```
            for (String str: s2) {
                s1.add(str);
            }


            return s1;
        }

        private static int priority(String option){
            if(option.equals("+") || option.equals("-"))
                return 1;
            else if(option.equals("("))
                return 3;
            else
                return 2;
        }

        private static ArrayList<String> getMidExpress(String expression) {
            ArrayList<String> list = new ArrayList<>(20);
            //遍历表达式
            //判断是否继续
            int lenth = expression.length();
            int i = 0;
            char cur = ' ';
            char next = ' ';
            String target ="";
            while(true){
                if(i == lenth)break;
                cur = expression.charAt(i);
                //若是符号则直接进入集合
                if(cur <48 || cur > 57){
                    list.add(cur+"");
                }else {
                    //若是数字则判断下一个是否为数字
                    while(true){
                    target+=cur;
                    if(i == lenth-1) break;
                    next=expression.charAt(i+1);
                    if(next < 48 || next > 57){
                        break;
                    }
                    i++;
                    cur = expression.charAt(i);
                    }
                    list.add(target);
                    target="";
                }
                i++;
            }

            return list;
        }

    }
```