Java实现带括号表达式的计算器

一、具体功能

1、：输入，输出

输入：允许输入带有括号的完整计算式（例 8*（4-95)+5÷2*e-pi）

输出：输出Double类型的结果

输出：整个运算表达式并保存于历史记录中

2、：功能

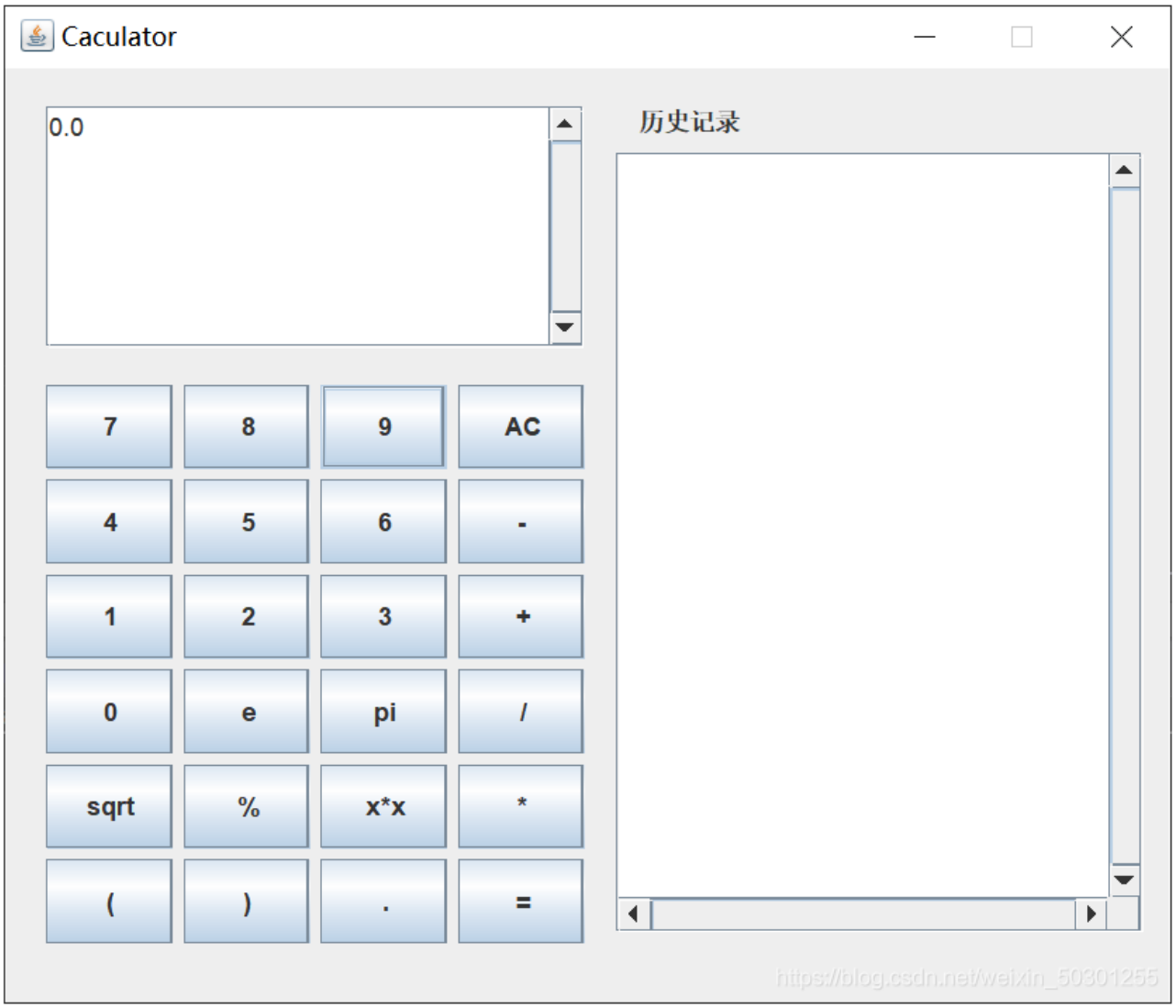
基本的加，减，乘，除，四则运算

平方运算

开方运算

求余运算

最终界面如下图：



除了常规的数字按钮和运算符，还有两个常数e，pi（π），清空键AC，括号运算符()，平方(x^x)和开方(sqrt)运算符，输入显示框以及历史记录文本框，文本框的垂直滚动条和水平滚动条。

二、主要思想

1：中缀表达式转为后缀表达式

准备：

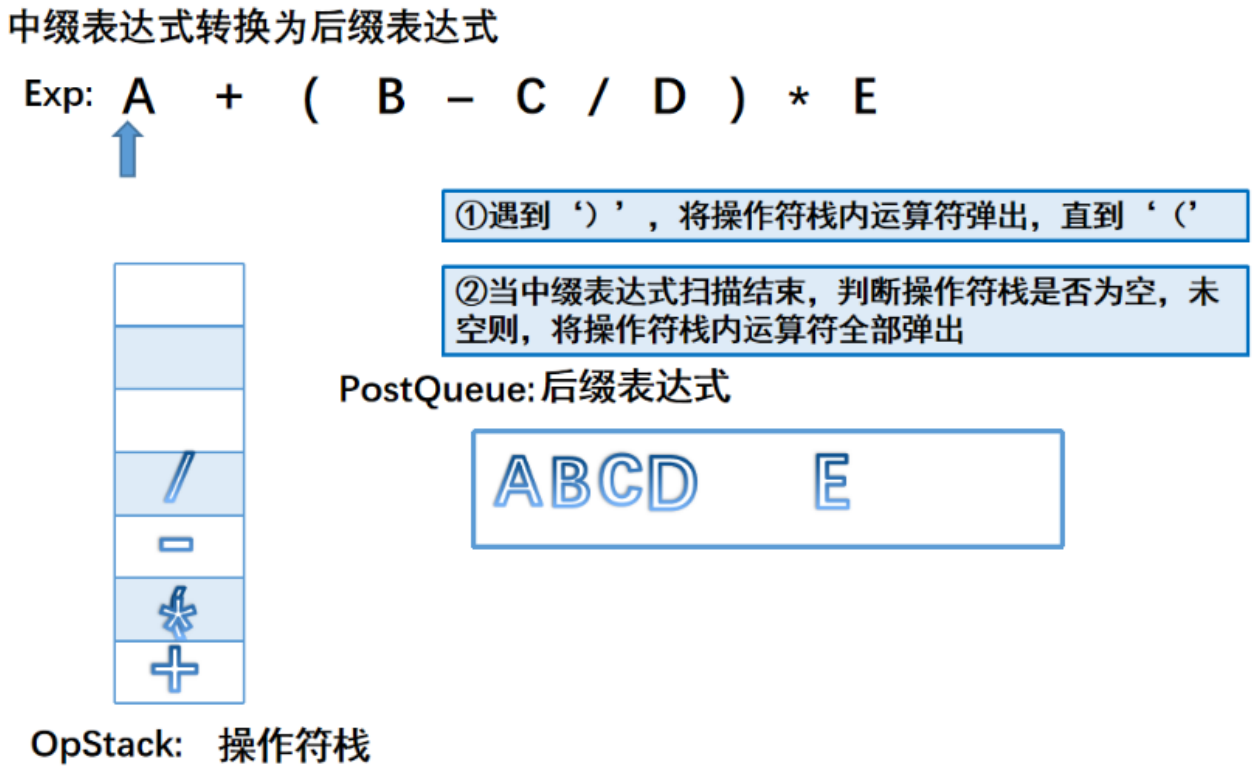
①后缀表达式队列：postQueue，用于存储逆波兰表达式（其实不用队列排序直接输出也行）
②操作符栈：opStack，对用户输入的操作符进行处理，用于存储运算符

算法思想：

从左向右依次读取算术表达式的元素X，分以下情况进行不同的处理：
（1）如果X是操作数，直接入队
（2）如果X是运算符，再分以下情况：
a）如果栈为空，直接入栈。
b）如果X==”（“，直接入栈。
c）如果X==”）“，则将栈里的元素逐个出栈，并入队到后缀表达式队列中，直到第一个配对的”(”出栈。（注：“(”和“)”都不 入队）

d）如果是其他操作符（+ – * /）,则和栈顶元素进行比较优先级。 如果栈顶元素的优先级大于等于X，则出栈并把栈中弹出的元素入队，直到栈顶元素的优先级小于X或者栈为空。弹出完这些元素后，才将遇到的操作符压入到栈中。
(3)最后将栈中剩余的操作符全部入队。

示意图：



2、计算后缀表达式

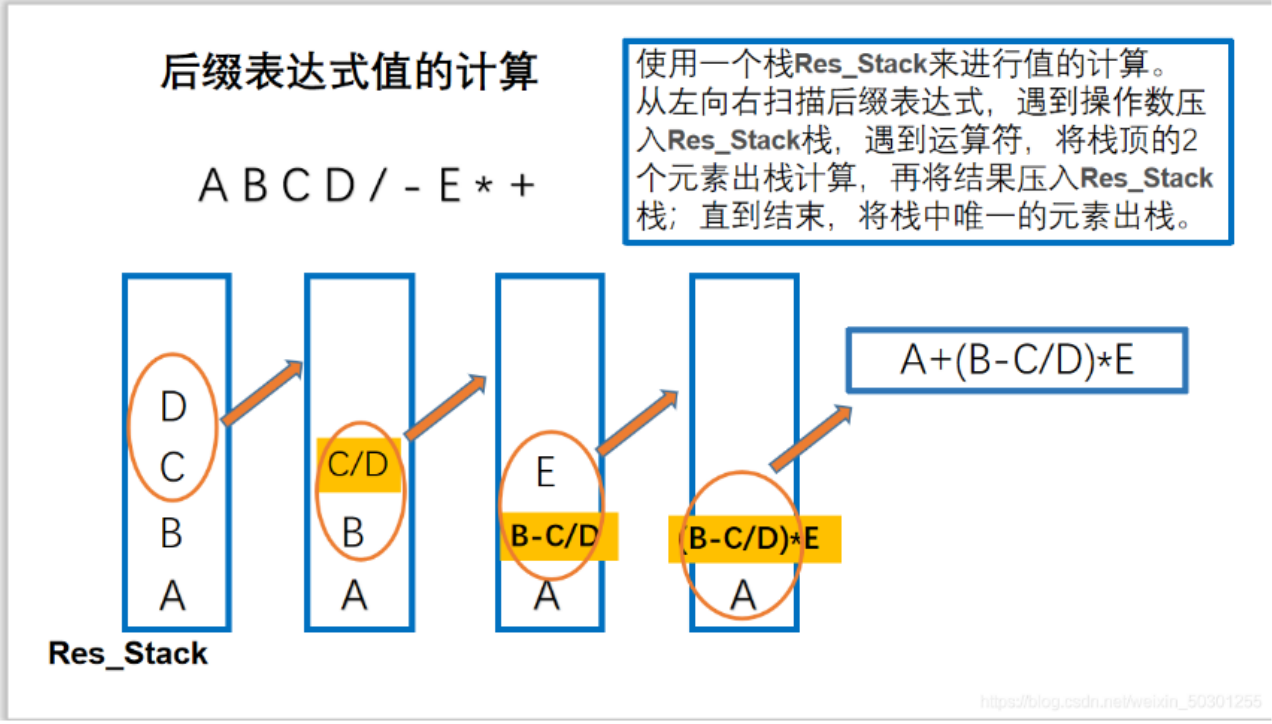
准备：

需要用到一个结果栈Res_Stack ： 用于存放计算的中间过程的值和最终结果

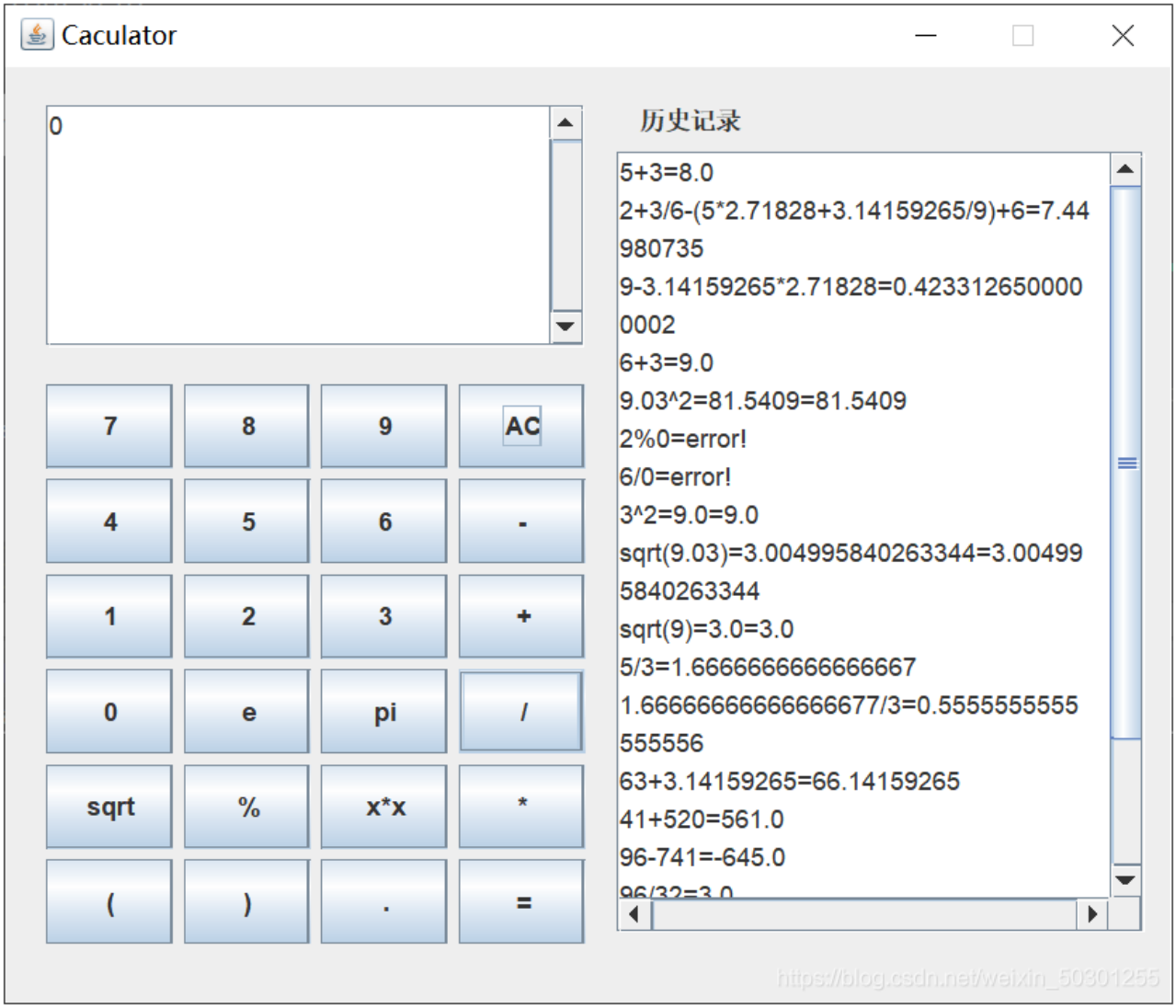
算法思想：

1、从左开始向右遍历后缀表达式的元素。
2、如果取到的元素是操作数，直接入栈Res_Stack，如果是运算符，从栈中弹出2个数进行运算，然后把运算结果入栈
3、当遍历完后缀表达式时，计算结果就保存在栈里了。

示意图：


后缀表达式值的计算
A B C D / - E * +
使用一个栈Res_Stack来进行值的计算。
从左向右扫描后缀表达式，遇到操作数压入Res_Stack栈，遇到运算符，将栈顶的2个元素出栈计算，再将结果压入Res_Stack栈；直到结束，将栈中唯一的元素出栈。
D
C
B
A
C/D
B
A
E
B-C/D
A
(B-C/D)*E
A
A+(B-C/D)*E
Res_Stack

三、结果测试



分析：

1、可实现基本四则运算及平方、开方、求余运算。
2、运算表达式可显示于输入界面并保存于历史记录栏
3、输入界面和历史记录栏皆可实现不断字自动换行功能以及滚动条功能
4、不足之处：进行平方和开方运算时其保存在历史记录中的表达式会出现两个等号及两个结果。

四、完整源代码(每行代码已附有详细注释)

```
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Objects;

import javax.swing.*;

//Calculator类，继承JFrame框架，实现事件监听器接口
public class Calculator extends JFrame implements ActionListener {
    private final String[] KEYS = { "7", "8", "9", "AC", "4", "5", "6", "-", "1", "2", "3", "+", "0", "e", "pi", "/", "sqrt",
            "%", "x*x", "*", "(", ")", ".", "=" };
    private JButton keys[] = new JButton[KEYS.length];
```

```
    private JTextArea resultText = new JTextArea("0.0");// 文本域组件TextArea可容纳多行文本；文本框内容初始值设为0.0
    private JTextArea History = new JTextArea();// 历史记录文本框初始值设为空
    private JPanel jp2=new JPanel();
    private JScrollPane gdt1=new JScrollPane(resultText);//给输入显示屏文本域新建一个垂直滚动滑条
    private JScrollPane gdt2=new JScrollPane(History);//给历史记录文本域新建一个垂直滚动滑条
    // private JScrollPane gdt3=new JScrollPane(History);//给历史记录文本域新建一个水平滚动滑条
    private JLabel label = new JLabel("历史记录");
    private String b = "";

    // 构造方法
    public Calculator() {
        super("Caculator");//"超"关键字，表示调用父类的构造函数，
        resultText.setBounds(20, 18, 255, 115);// 设置文本框大小
        resultText.setAlignmentX(RIGHT_ALIGNMENT);// 文本框内容右对齐
        resultText.setEditable(false);// 文本框不允许修改结果
        History.setBounds(290, 40, 250,370);// 设置文本框大小
        History.setAlignmentX(LEFT_ALIGNMENT);// 文本框内容右对齐
        History.setEditable(false);// 文本框不允许修改结果
        label.setBounds(300, 15, 100, 20);//设置标签位置及大小
        jp2.setBounds(290,40,250,370);//设置面板窗口位置及大小
        jp2.setLayout(new GridLayout());
        JPanel jp1 = new JPanel();
        jp1.setBounds(20,18,255,115);//设置面板窗口位置及大小
        jp1.setLayout(new GridLayout());
        resultText.setLineWrap(true);// 激活自动换行功能
        resultText.setWrapStyleWord(true);// 激活断行不断字功能
        resultText.setSelectedTextColor(Color.RED);
        History.setLineWrap(true);//自动换行
        History.setWrapStyleWord(true);
        History.setSelectedTextColor(Color.blue);
        gdt1.setViewportView(resultText);//使滚动条显示出来
        gdt2.setViewportView(History);
        gdt1.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);//设置让垂直滚动条一直显示
        gdt2.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);//设置让垂直滚动条一直显示
        gdt2.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);//设置让水平滚动条一直显示
        jp1.add(gdt1);//将滚动条添加入面板窗口中
        jp2.add(gdt2);
        this.add(jp1);//将面板添加到总窗体中
        this.add(jp2);//将面板添加到总窗体中
        this.setLayout(null);
        this.add(label);// 新建"历史记录"标签
        //this.add(resultText);// 新建文本框，该语句会添加进去一个新的JTextArea导致带有滚动条的文本无法显示或者发生覆
盖
        //this.add(History);// 新建历史记录文本框，该语句会添加进去一个新的JTextArea导致带有滚动条的文本无法显示

        // 放置按钮
        int x = 20, y = 150;
        for (int i = 0; i < KEYS.length; i++)
        {
            keys[i] = new JButton();
            keys[i].setText(KEYS[i]);
            keys[i].setBounds(x, y, 60, 40);
            if (x < 215) {
                x += 65;
            } else {
                x = 20;
                y += 45;
            }
```

```
            this.add(keys[i]);
        }
        for (int i = 0; i < KEYS.length; i++)// 每个按钮都注册事件监听器
        {
            keys[i].addActionListener(this);
        }
        this.setResizable(false);
        this.setBounds(500, 200, 567, 480);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
    }

    // 事件处理
    public void actionPerformed(ActionEvent e)
    {
        //History.setText(b);//使输入的表达式显示在历史记录文本框中
        String label=e.getActionCommand();//获得事件源的标签
        if(Objects.equals(label, "="))//
        {
            resultText.setText(this.b);
            History.setText(History.getText()+resultText.getText());
            if(label.equals("="))//调用计算方法，得出最终结果
            {
                String[] s =houzhui(this.b);
                String result=Result(s);
                this.b=result+"";
                //更新文本框，当前结果在字符串b中，并未删除，下一次输入接着此结果以实现连续运算
                resultText.setText(this.b);
                History.setText(History.getText()+"="+resultText.getText()+"\n");
            }
        }
        else if(Objects.equals(label, "AC"))//清空按钮，消除显示屏文本框前面所有的输入和结果
        {
            this.b="";
            resultText.setText("0");//更新文本域的显示，显示初始值;
        }
        else if(Objects.equals(label, "sqrt"))
        {
            String n=kfys(this.b);
            resultText.setText("sqrt"+"("+this.b+")"+"="+n);//使运算表达式显示在输入界面
            History.setText(History.getText()+"sqrt"+"("+this.b+")"+"=");//获取输入界面的运算表达式并使其显示在历史记录文
本框
            this.b=n;
        }
        else if(Objects.equals(label, "x*x"))
        {
            String m=pfys(this.b);
            resultText.setText(this.b+"^2"+"="+m);//使运算表达式显示在输入界面
            History.setText(History.getText()+this.b+"^2"+"=");//获取输入界面的运算表达式并使其显示在历史记录文本框
            this.b=m;
        }
        else if(Objects.equals(label, "e") || Objects.equals(label, "pi"))
        {
            if(label.equals("e"))
            {
                String m=String.valueOf(2.71828);//将e的值以字符串的形式传给m
                this.b=this.b+m;//保留显示m之前输入的运算符或数字字符继续下一步运算
                resultText.setText(this.b);
```

```
            // History.setText(History.getText()+this.b);
        }
        if(label.equals("pi"))
        {
            String m=String.valueOf(3.14159265);
            this.b=this.b+m;
            resultText.setText(this.b);
            // History.setText(History.getText()+this.b);
        }
    }
    else
    {
        this.b=this.b+label;
        resultText.setText(this.b);
        // History.setText(History.getText()+this.b);


    }
    //History.setText(History.getText()+this.b);//使输入的表达式显示在历史记录文本框中
}
//将中缀表达式转换为后缀表达式
private String[] houzhui(String str) {
    String s = "";// 用于承接多位数的字符串
    char[] opStack = new char[100];// 静态栈,对用户输入的操作符进行处理，用于存储运算符
    String[] postQueue = new String[100];// 后缀表达式字符串数组，为了将多位数存储为独立的字符串
    int top = -1, j = 0;// 静态指针top,控制变量j
    for (int i = 0; i < str.length(); i++)// 遍历中缀表达式
    // indexof函数，返回字串首次出现的位置；charAt函数返回index位置处的字符；
    {
        if ("0123456789.".indexOf(str.charAt(i)) >= 0) // 遇到数字字符的情况直接入队
        {
            s = "";// 作为承接字符，每次开始时都要清空
            for (; i < str.length() && "0123456789.".indexOf(str.charAt(i)) >= 0; i++) {
                s = s + str.charAt(i);
                //比如，中缀表达式：234+4*2，我们扫描这个字符串的时候，s的作用相当于用来存储长度为3个字符的操作
数：234
            }
            i--;
            postQueue[j] = s;// 数字字符直接加入后缀表达式
            j++;
        }
        else if ("(".indexOf(str.charAt(i)) >= 0) {// 遇到左括号
            top++;
            opStack[top] = str.charAt(i);// 左括号入栈
        }
        else if (")".indexOf(str.charAt(i)) >= 0) {// 遇到右括号
            for (;;)// 栈顶元素循环出栈，直到遇到左括号为止
            {
                if (opStack[top] != '(') {// 栈顶元素不是左括号
                    postQueue[j] = opStack[top] + "";// 栈顶元素出栈
                    j++;
                    top--;
                } else { // 找到栈顶元素是左括号
                    top--;// 删除栈顶左括号
                    break;// 循环结束
                }
            }
        }
```

```
            else if ("*%/+-".indexOf(str.charAt(i)) >= 0)// 遇到运算符
            {
                if (top == -1)
                 {// 若栈为空则直接入栈
                    top++;
                    opStack[top] = str.charAt(i);
                }
                else if ("*%/".indexOf(opStack[top]) >= 0)
                {// 当栈顶元素为高优先级运算符时,让栈顶元素出栈进入后缀表达式后,当前运算符再入栈
                        postQueue[j] = opStack[top] + "";
                        j++;
                    opStack[top] = str.charAt(i);
                }
                else
                {
                    top++;
                    opStack[top] = str.charAt(i);// 当前元素入栈
                }
            }
        }
        while (top != -1) {// 遍历结束后将栈中剩余元素依次出栈进入后缀表达式
            postQueue[j] = opStack[top] + "";
            j++;
            top--;
        }
        return postQueue;
    }

    //开方运算方法
    public String kfys(String str) {
        String result = "";
        double a = Double.parseDouble(str), b = 0;
        b = Math.sqrt(a);
        result = String.valueOf(b);//将运算结果转换为string类型并赋给string类型的变量result
        return result;
    }

    //平方运算方法
    public String pfys(String str) {
        String result = "";
        double a = Double.parseDouble(str), b = 0;
        b = Math.pow(a, 2);
        result = String.valueOf(b);
        return result;
    }

    // 计算后缀表达式，并返回最终结果
    public String Result(String[] str) {
        String[] Result = new String[100];// 顺序存储的栈，数据类型为字符串
        int Top = -1;// 静态指针Top
        for (int i = 0; str[i] != null; i++) {
            if ("+-*%/".indexOf(str[i]) < 0) {  //遇到数字，直接入栈
                Top++;
                Result[Top] = str[i];
            }
            if ("+-*%/".indexOf(str[i]) >= 0)// 遇到运算符字符，将栈顶两个元素出栈计算并将结果返回栈顶
            {
                double x, y, n;
```

```
                x = Double.parseDouble(Result[Top]);// 顺序出栈两个数字字符串，并转换为double类型
                Top--;
                y = Double.parseDouble(Result[Top]);
                Top--;
                if ("*".indexOf(str[i]) >= 0) {
                    n = y * x;
                    Top++;
                    Result[Top] = String.valueOf(n);// 将运算结果重新入栈

                }
                if ("/".indexOf(str[i]) >= 0)
                {
                    if (x == 0)// 被除数不允许为0
                    {
                        String s = "error!";
                        return s;
                    } else {
                        n = y / x;
                        Top++;
                        Result[Top] = String.valueOf(n);// 将运算结果重新入栈
                    }
                }
                if ("%".indexOf(str[i]) >= 0)
                {
                    if (x == 0)// 被除数不允许为0
                    {
                        String s = "error!";
                        return s;
                    } else {
                        n = y % x;
                        Top++;
                        Result[Top] = String.valueOf(n);// 将运算结果重新入栈
                    }
                }
                if ("-".indexOf(str[i]) >= 0) {
                    n = y - x;
                    Top++;
                    Result[Top] = String.valueOf(n);// 将运算结果重新入栈
                }
                if ("+".indexOf(str[i]) >= 0) {
                    n = y + x;
                    Top++;
                    Result[Top] = String.valueOf(n);// 将运算结果重新入栈
                }
            }
        }
        return Result[Top];// 返回最终结果
    }

    // 主函数
    public static void main(String[] args) {
        Calculator a = new Calculator();
    }
}
```