简述

这23个设计模式中最难的就是解释器模式了，在实际开发过程中也很少会用到这个模式，JAVA 中如果碰到可以用 expression4J 代替

给定一个语言，定义它的文法表示，并定义一个解释器，这个解释器使用该标识来解释语言中的句子

角色

抽象解释器（AbstractExpression）
具体的解释任务由各个实现类完成

终结符表达式（TerminalExpression）
实现与文法中的元素相关联的解释操作，通常一个解释器模式中只有一个终结表达式，但有多个实例，对应不同的终结符

非终结符表达式（NonterminalExpression）
文法中的每条规则对应于一个非终结表达式，非终结符表达式根据逻辑的复杂程度而增加，原则上每个文法规则都对应一个非终结符表达式

上下文（Context）
上下文环境类,包含解释器之外的全局信息

客户类（Test）
客户端,解析表达式,构建抽象语法树,执行具体的解释操作等.

代码

```
/**
 * 解释器接口
 */
public interface Expression {
    int interpreter(Context context);//一定会有解释方法
}

```

```java
import java.util.HashMap;
import java.util.Map;
import java.util.Stack;

/**
 *  上下文类（这里主要用来将变量解析成数字【当然一开始要先定义】）
 */
public class Context {
    private Map<Expression, Integer> map = new HashMap<>();

    public void add(Expression s, Integer value){
        map.put(s, value);
    }
    public Integer lookup(Expression s){
        return map.get(s);
    }
    //构建语法树的主要方法
    public static Expression build(String str) {
        //主要利用栈来实现
        Stack<Expression> objects = new Stack<>();
        for (int i = 0; i < str.length(); i++){
            char c = str.charAt(i);
            //遇到运算符号+号时候
            if (c == '+'){

                //先出栈
                Expression pop = objects.pop();

                //将运算结果入栈
                objects.push(new PlusOperation(pop, new TerminalExpression(String.valueOf(str.charA
            } else if (c == '-'){
                //遇到减号类似加号
                Expression pop = objects.pop();

                objects.push(new MinusOperation(pop, new TerminalExpression(String.valueOf(str.char

            } else {
                //遇到非终结符直接入栈（基本就是第一个数字的情况）
                objects.push(new TerminalExpression(String.valueOf(str.charAt(i))));
            }
        }
        //把最后的栈顶元素返回
        return objects.pop();
    }
}


/**
 *   抽象非终结符表达式
 */
public abstract class NonTerminalExpression implements Expression{

    Expression e1,e2;

```

```
63      public NonTerminalExpression(Expression e1, Expression e2){
64          this.e1 = e1;
65          this.e2 = e2;
66      }
67  }
68
69  /**
70   *  终结符表达式（在这个例子，用来存放数字，或者代表数字的字符）
71   */
72  public class TerminalExpression implements Expression{
73
74
75      String variable;
76      public TerminalExpression(String variable){
77
78          this.variable = variable;
79      }
80      @Override
81      public int interpreter(Context context) {
82          //因为要兼容之前的版本
83          Integer lookup = context.lookup(this);
84          if (lookup == null)
85              //若在map中能找到对应的数则返回
86              return Integer.valueOf(variable);
87          //找不到则直接返回（认为输入的就是数字）
88          return lookup;
89      }
90  }
91
92
93  /**
94   *  加法表达式实现类
95   */
96  public class PlusOperation extends NonTerminalExpression {
97
98      public PlusOperation(Expression e1, Expression e2) {
99          super(e1, e2);
100      }
101
102      //将两个表达式相加
103      @Override
104      public int interpreter(Context context) {
105          return this.e1.interpreter(context) + this.e2.interpreter(context);
106      }
107  }
108
109
110  /**
111   *  减法表达式实现类
112   */
113  public class MinusOperation extends NonTerminalExpression {
114
115      public MinusOperation(Expression e1, Expression e2) {
116          super(e1, e2);
```

```
    }

    //将两个表达式相减
    @Override
    public int interpreter(Context context) {
        return this.e1.interpreter(context) - this.e2.interpreter(context);
    }
}


public class Main {
    public static void main(String[] args) {

        Context context = new Context();
        TerminalExpression a = new TerminalExpression("a");
        TerminalExpression b = new TerminalExpression("b");
        TerminalExpression c = new TerminalExpression("c");
        String str = "4+8-2+9+9-8";
        Expression build = Context.build(str);
        System.out.println("4+8-2+9+9-8=" + build.interpreter(context));

        context.add(a, 4);
        context.add(b, 8);
        context.add(c, 2);

        System.out.println(new MinusOperation(new PlusOperation(a,b), c).interpreter(context));
    }
}
```