

合肥工业大学

系统软件综合设计报告

编译原理分册

设计题目 将算术表达式转换成三元式的程序实现

学生姓名 林天岳

学 号 2017217893

专业班级 计算机科学与技术 17-5

指导教师 唐益明

完成日期 2020-8-24

一、设计目的及设计要求

设计一个语法制导翻译器，将算术表达式翻译成三元式。

要求：先确定一个定义算术表达式的文法，为其设计一个语法分析程序，为每条产生式配备一个语义子程序，按照一遍扫描的语法制导翻译方法，实现翻译程序。对用户输入的任意一个正确的算术表达式，程序将其转换成三元式输出

二、开发环境描述

算法部分 java8 idea ultimate 2020

界面部分 React.JS + materialUI + Echarts.JS

三、设计内容、主要算法描述

主要思想：首先对用户输入语句进行词法分析，存入对应数据结构，在对其进行 LR1 语法分析，再根据分析栈获取中间表达式，再根据由语义子程序递归生成对应语法树。

四、设计的输入和输出形式

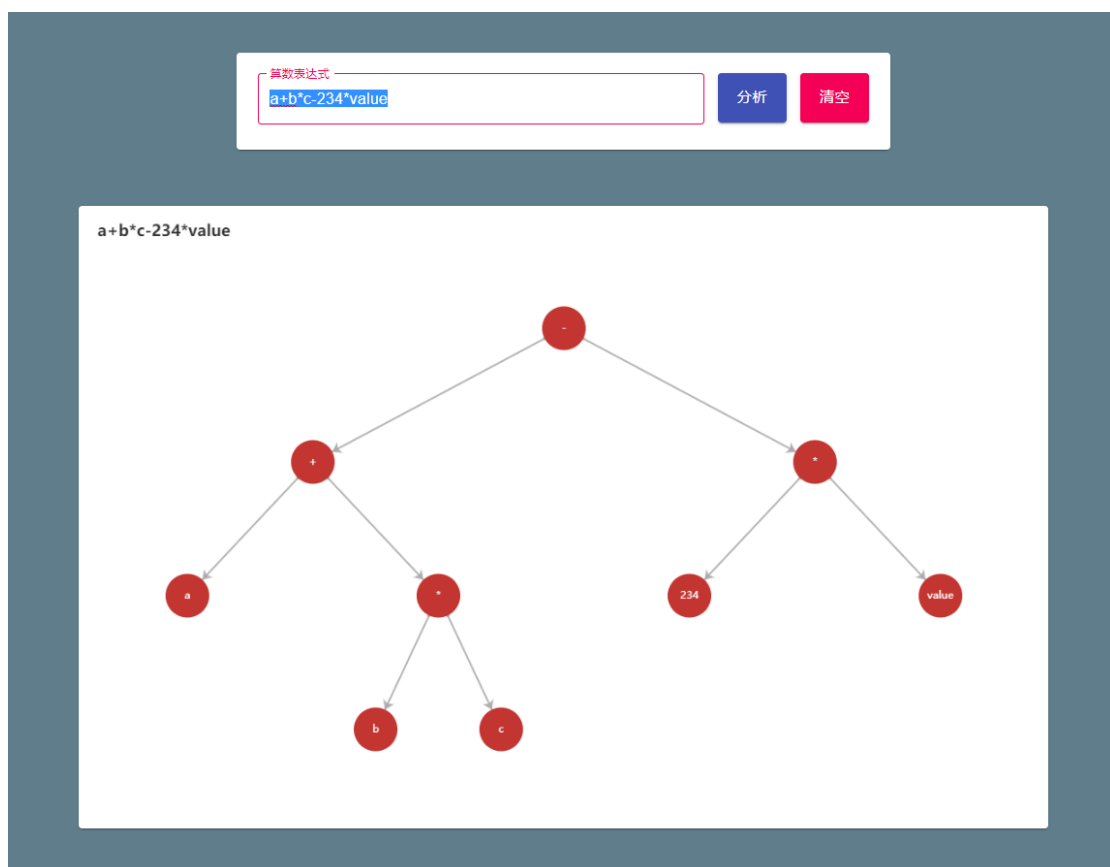
输入：合法的算数表达式，可包含变量

输出：JSON 格式+语法树

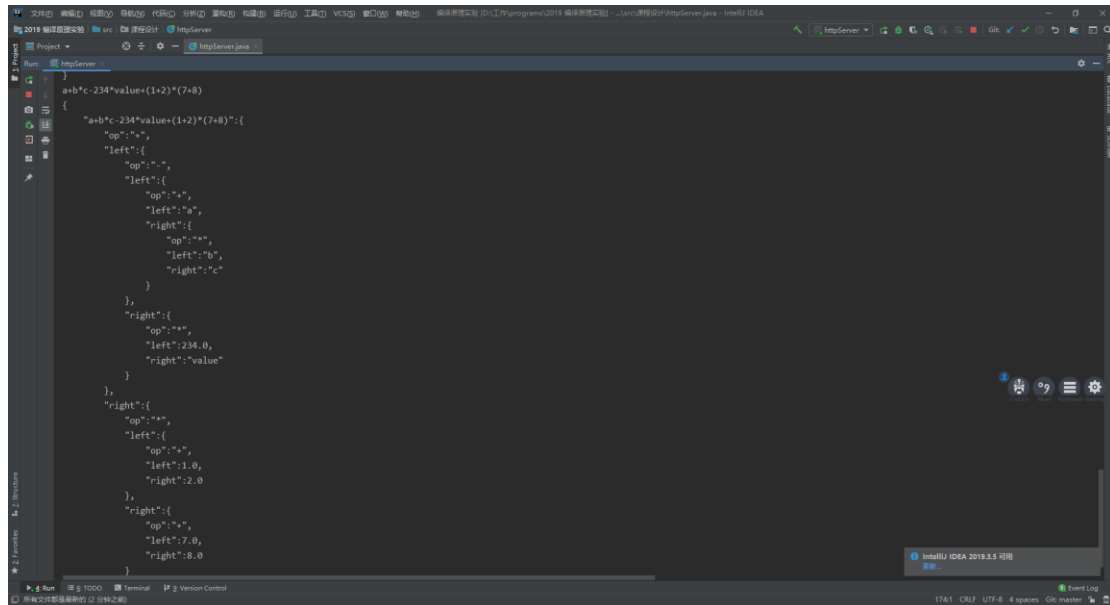
例如：输入 $a+b*c-234*value$

输出：

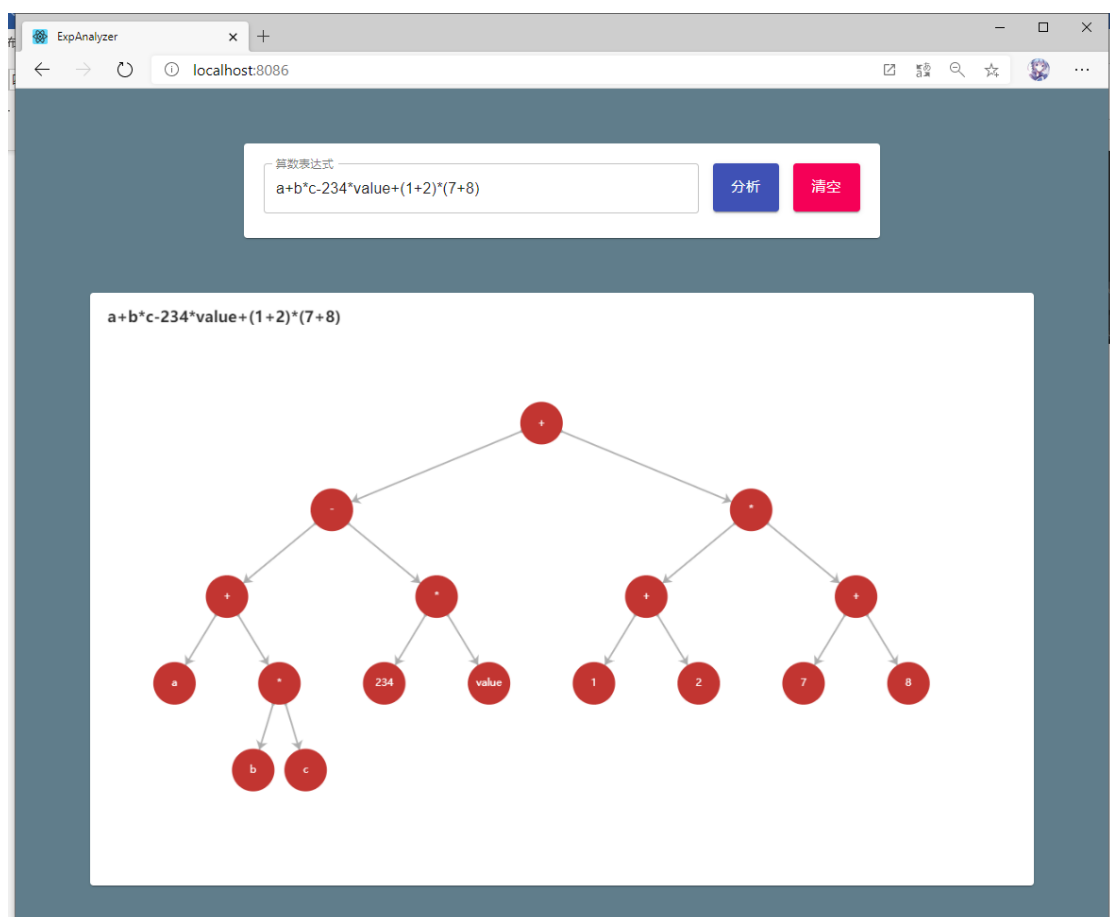
```
a+b*c-234*value
{
  "a+b*c-234*value":{
    "op":"-",
    "left":{
      "op":"+",
      "left":"a",
      "right":{
        "op":"*",
        "left":"b",
        "right":"c"
      }
    },
    "right":{
      "op":"*",
      "left":234.0,
      "right":"value"
    }
  }
}
```



五、程序运行的结果



```
1 }
2 a+b*c-234*value+(1+2)*(7+8)
3 {
4   "a+b*c-234*value+(1+2)*(7+8)": {
5     "op": "+",
6     "left": {
7       "op": "-",
8       "left": {
9         "op": "+",
10        "left": "a",
11        "right": {
12          "op": "*",
13          "left": "b",
14          "right": "c"
15        }
16      },
17      "right": {
18        "op": "*",
19        "left": "234.0",
20        "right": "value"
21      }
22    },
23    "right": {
24      "op": "+",
25      "left": {
26        "op": "+",
27        "left": "1.0",
28        "right": "2.0"
29      },
30      "right": {
31        "op": "+",
32        "left": "7.0",
33        "right": "8.0"
34      }
35    }
36  }
37 }
```










六、总结

本次实验是建立在之前实验的基础上的。由于词法分析器和 LR1 分析器做的比较完善，所以简单修改就可以直接使用。设计 cell 结构，保存类型与原始数据信息，数值类型 toString() 返回类型，算符返回符号，生成 RL1 直接识别的结构。如果输入表达式正确无误，则可以得到语法分析栈。此时再使用对应的语法分析程序，对于每一条产生式，递归生成对应语法树。

界面部分，主要是对于语法树的渲染展示。使用对应的算法，确定位置并使用 Echart.JS 展示。

七、源程序清单作为报告的附件。

	GUI	2020/8/29 18:38	文件夹	
	httpServer.java	2020/8/25 8:33	Java 源文件	3 KB
	LexicalAna.java	2020/2/10 15:57	Java 源文件	9 KB
	LR1.java	2020/8/25 8:48	Java 源文件	26 KB
	treeBuilder.java	2020/8/24 14:37	Java 源文件	9 KB
	App.js	2020/8/24 14:49	JavaScript 源文件	8 KB
	Chart.js	2020/8/24 13:25	JavaScript 源文件	2 KB

全部代码已上传 <https://github.com/DriverLin/Compilation-principle>

以下为 cell 生成和语法分析程序部分代码

```
1. package 课程设计;
2.
3. import java.beans.Bean;
4. import java.io.*;
5. import java.net.HttpURLConnection;
6. import java.net.URL;
7. import java.net.http.HttpRequest;
8. import java.nio.charset.StandardCharsets;
9. import java.nio.file.Files;
10. import java.nio.file.Paths;
11. import java.util.*;
12. import java.util.regex.Pattern;
13.
```

```

14. import com.alibaba.fastjson.JSON;
15. import com.alibaba.fastjson.JSONArray;
16. import com.alibaba.fastjson.annotation.JSONField;
17. import com.alibaba.fastjson.serializer.SerializerFeature;
18.
19. public class treeBuilder {
20.     public String getResult(String arg) throws IOException {
21.         analyser analyser = new analyser("S`->E\n" +
22.             "E->E+T | E-T|T\n" +
23.             "T->T*F | T/F|F\n" +
24.             "F->(E) | i");
25.         LinkedHashMap res = new LinkedHashMap();
26.         res.put(arg,analyser.ana(arg));
27.         String beautify = JSON.toJSONString(res, SerializerFeature.PrettyFor
            mat, SerializerFeature.WriteMapNullValue,
28.             SerializerFeature.WriteDateUseDateFormat);
29.         return beautify;
30.
31.     }
32. }
33.
34. class analyser{
35.     LR1 sol;
36.     public analyser(String rule){
37.         sol = new LR1(rule);
38.     }
39.     public Object ana(String exp){
40.         Stack res = sol.analyse(new Express(InitExp(exp)));
41.         if ( !((Map)res.pop()).get("action").equals("acc")) {
42.             return "ExpressionError";
43.         }
44.         Node head = new Node(new Cell("s","E"));
45.         getNodes(res,head);
46.         simplify(head);
47.         return simplePrintNode(head);
48.     }
49.     private List<Cell> InitExp_2(String exp) {
50.         List<Cell> result = new ArrayList<>();
51.         List<String> symbols = Arrays.asList("\\+", "-
", "\\*", "/", "\\(", "\\)");
52.         for (String sym : symbols) {
53.             exp = exp.replaceAll(sym,"|"+sym+"|");
54.         }
55.         exp = exp.replaceAll("\\|\\\\\\\\|","|");

```

```

56.         for (String s : exp.split("\\|")) { //切片问题  ()在首部会导致切片问
           题 |(|123|+|456|)| -> ["", "(", "123", "+", "456", ")"]
57.             if(s.equals("")){
58.                 continue;
59.             }
60.             if("+-*/()".contains(s)){
61.                 result.add(new Cell("s",s));
62.             }
63.             else{
64.                 result.add(new Cell("i",s));
65.             }
66.         }
67.         return result;
68.     }
69.     private List<Cell> InitExp(String exp){
70.         List cells = new ArrayList();
71.         LexicalAna lexicalAna = new LexicalAna();
72.         for (Result result : lexicalAna.Solve(new String[]{exp})) {
73.             if(result.kind.equals("常数") || result.kind.equals("标识符
           ") ){
74.                 cells.add(new Cell("i",result.word));
75.             }
76.             else {
77.                 cells.add(new Cell("s",result.word));
78.             }
79.         }
80.         return cells;
81.     }
82.     private void getNodes(Stack processRec,Node head){
83.         if(processRec.empty()){
84.             return;
85.         }
86.         if(head == null){
87.             return;
88.         }
89.         Map rec = (Map)processRec.pop();
90.         Object action = rec.get("action");
91.         if(action.getClass().equals(new Cell().getClass())){
92.             Cell cell = (Cell)action;
93.             //         System.out.println("回填"+cell.getValue());
94.             if(cell.getType().equals("i")){//是数据而不是()
95.                 head.cell = cell;//数据回填
96.             }
97.         }

```

```

98.         else {
99.             String act = (String)action;
100.             String [] expRight = act.split("->")[1].split("");
101.             if(expRight.length == 3){ //运算操作
102.                 head.left = new Node(new Cell("s",expRight[0]));
103.                 head.center = new Node(new Cell("s",expRight[1]));
104.                 head.right = new Node(new Cell("s",expRight[2]));
105.             }
106.             else {//变换操作
107.                 head.left = null;
108.                 head.right = null;
109.                 head.center = new Node(new Cell("s",expRight[0]));
110.             }
111.         }
112.         getNodes(processRec,head.right);
113.         getNodes(processRec,head.center);
114.         getNodes(processRec,head.left);
115.         return;
116.     }
117.     private String printNode(Node head){
118.         if(head == null) {
119.             return "null";
120.         }
121.         return String.format("{\\\"head\\\":\\\"%s\\\",\\\"left\\\":%s,\\\"center\\\":%s,\\\"right\\\":%s}",head.cell.getValue(),printNode(head.left),printNode(head.center),printNode(head.right) );
122.     }
123.     private void simplify(Node head){
124.         if(head == null){
125.             return;
126.         }
127.         if(head.left == null && head.right == null && head.center != null)
128.         {
129.             head.cell = head.center.cell;
130.             head.left = head.center.left;
131.             head.right = head.center.right;
132.             head.center = head.center.center;//顺序
133.         }
134.         if(head.left!=null&&head.left.cell.getValue().equals("(")){
135.             head.left = null;
136.         }
137.         if(head.right!=null&&head.right.cell.getValue().equals(")")){
138.             head.right = null;
139.         }

```



```

139.         simplify(head.left);
140.         simplify(head.center);
141.         simplify(head.right);
142.     }
143.     private Object simplePrintNode(Node head){
144.         if( head.left != null && head.right != null && head.center !=null){

145.             Data data = new Data();
146.             data.setOp(simplePrintNode(head.center));
147.             data.setLeft(simplePrintNode(head.left));
148.             data.setRight(simplePrintNode(head.right));
149.             return data;
150.         }
151.         if( head.left == null && head.right == null && head.center!=null){

152.             return simplePrintNode(head.center);
153.         }
154.         if(head.left == null && head.center == null && head.right == null){

155.             String value = head.cell.getValue();
156.             if(head.cell.getType().equals("i")){//数字或者未知数
157.                 if(Pattern.matches("(-?\d*)\\.?\d+", head.cell.getValue()
158. )){
159.                     return value.contains(".")? Double.parseDouble(value) :
160. Integer.parseInt(value) ;
161.                 }else {
162.                     return value;
163.                 }
164.             }else {
165.                 return value;//运算符
166.             }
167.         }
168.         return null;
169.     }
170.
171.     class Cell {
172.         String type;
173.         String value;
174.         int index;
175.         static int count = 0;
176.         Cell(){
177.             this.type = null;

```

```

178.         this.value = null;
179.         this.index = -1;
180.     }
181.     Cell(String type, String value){
182.         this.type = type;
183.         this.value = value;
184.         if( this.type.equals("i") ){
185.             index = count;
186.             count++;
187.         }
188.         else{
189.             index = -1;
190.         }
191.     }
192.     @Override
193.     public String toString() {
194.         return type.equals("i")? type : value;
195.     }
196.     public String getValue() {
197.         return value;
198.     }
199.     public String getType() {
200.         return type;
201.     }
202. }
203.
204. class Express{
205.     List<Cell> cells;
206.     public Express(List<Cell> cells){
207.         this.cells = cells;
208.     }
209.     @Override
210.     public String toString() {
211.         StringBuffer sb = new StringBuffer();
212.         cells.forEach(N -> sb.append(N));
213.         return sb.toString();
214.     }
215.     public List<Cell> getCells() {
216.         return cells;
217.     }
218. }
219.
220. class Node{
221.     Node left;

```

```
222.     Node center;
223.     Node right;
224.     Cell cell;
225.     public Node(Cell value){
226.         this.cell = value;
227.         left = null;
228.         center = null;
229.         right = null;
230.     }
231.     @Override
232.     public String toString() {
233.         return cell.toString();
234.     }
235. }
236.
237. @JavaBean
238. class Data{
239.     @JSONField(ordinal = 0)
240.     private Object op;
241.     @JSONField(ordinal = 1)
242.     private Object left;
243.     @JSONField(ordinal = 2)
244.     private Object right;
245.     public void setOp(Object op) {
246.         this.op = op;
247.     }
248.     public void setLeft(Object left) {
249.         this.left = left;
250.     }
251.     public void setRight(Object right) {
252.         this.right = right;
253.     }
254.     public Object getOp() {
255.         return op;
256.     }
257.     public Object getLeft() {
258.         return left;
259.     }
260.     public Object getRight() {
261.         return right;
262.     }
263.     @Override
264.     public int hashCode() {
```

```
265.         return (op.hashCode() + "/" + left.hashCode() + "/" + right.hashCode()
266.             + e()).hashCode();
267.     }
```