编译原理课程实验

实验报告

SimpleJavaLexerAndParser 实验报告

2021 年 1 月 20 日

71118313 张晓铮

东南大学软件学院

Contents

1 实验目标

本次实验的目标为开发一个利用软编码的方式，可以解析用户指定格式文件的 Lexer 和 Parser，其中对于词法和语法的 DFA 和 LR Table 由用户自己生成，本程序只根据用户输入的 DFA 和 LR Table 进行相应的解析

2 实验环境

- 本次实验采用的实验环境：
  1 操作系统：Windows 10
  2 语言：Java 14

3 实验内容

- 主要内容
  1 读取用户输入的 DFA 和 LR Table，并利用合适的数据结构存储起来
  2 编写读入源文件以及根据 DFA 和 LR Table 解析的程序

4 实验实现方法

在本次实验中，由于 DFA 和 LR Table 最终都可以用一张表来表示，所以采用 csv 文件来存储用户输入的 LR Table 和 DFA，以实现软编码的目的。然后根据读入的表进行解析，首先用 Lexer 读入用户输入的源文件，一次读入一个字符，并在 DFA 上进行状态之间的转换，并适当的进行报错处理。Lexer 提供了 next 函数，每次返回一个 Token 类，用来表示识别到的词，然后交由 Parser 处理，Parser 根据传入的 Token 和 LR Table 进行状态的移动，最终形成一棵 Parser Tree

4.1 SoftCodeLexer 关键内容描述

4.1.1 State

该类主要内容为 String name，boolean isAcceptState 以及 HashMap<String, String> mapping2statename; 它们分别记录了当前状态的名称，是否为结束节点，以及该状态能经过什么符号到达下一个状态的名称；

并且提供了方法 getNextStateName，它能根据传入的字符寻找能到达的状态的名称并返回，如果没有则返回 null

4.1.2 DFA

该类主要保存用户输入的 DFA,其中关键的数据成员为:HashMap<String, State> name2state，该 map 用于记录所有的 State 并将它们的名称和状态做映射；State start 用于记录开始状态；State currentState 用于 DFA 在状态之间转换时记录当前所在的状态；

该类提供了 next 函数，它接受一个字符，并判断能否从当前状态经过这个字符到达另一个状态，如果不可以则返回 false，可以的话返回 true；同时提供了函数 isAcceptingState，用来判断当前状态是否是 accept 的，还提供了 returnToStart 函数，用于将 DFA 的当前状态重置到开始状态。

对 DFA 做这样的封装，可以让外部使用者只调用 next，判断是否是 accepting 的即可，对于 DFA 内部的状态是如何转换的可以不关心。

4.1.3 Token

该类用于表示词法分析器生成的 Token，主要含有 Long id, String type, String lexValue，分别表示该 Token 的 id，类型和对应的解析出来的真实值。其中 type 为用户在配置文件中自己定义的名称。

4.1.4 Lexer

该类为实现词法分析的类，它使用封装好的 DFA，并维护了一个 char[] buf，buffer 的大小可由用户配置，其工作流程为每次读入 buffer 大小的字符，对字符进行遍历，遍历一个字符将其交给 DFA，如果 DFA 返回 false，说明不能继续走，然后判断 DFA 当前状态能否接受，如果能则输出，不能则报错。

4.2 SoftCodeParser 关键内容描述

4.2.1 Expression

该类主要用于存储用户输入的语法产生式，由于我们要求的产生式为上下文无关文法，所以该类用一个 String leftPart 存储产生式的左部，而右部用 ArrayList<String> rightPart 来按照顺序存储

4.2.2 Grammar

该类用于存储所有的产生式，它使用一个 HashMap<String, Expression> allExpression 将用户定义的产生式名称和产生式相关联并存储，并提供了按照名称返回产生式的方法。

4.2.3 LRTableRow

该类用于存储 LR Table 中的一行，类似于 DFA 中的状态，它主要含有两个数据成员：HashMap<String, Pair<String, Integer» actions，用来存储 action 表，它将表示该状态能从哪种 Token 转移到下一个状态或用什么来规约，Pair<String, Integer> 种用 Integer 表示是移进还是规约还是到达了 accept 状态，如果是移进，则 Pair 中第一个 String 表示要移进的状态名，如果是规约或者 accept 则表示产生式的名称；HashMap<String, String> goto 表示 goto 表，表示能从哪个非终结状态到达哪个状态；String name 代表当前行的状态名称。

4.2.4 LRTable

该类用来存储完整的 LRTable，利用 HashMap<String, LRTableRow> rows 存储所有行，并将每行的状态的名称和该行做映射，并提供了 canReach 和 canGoto 函数用来判断通过给定的终结符或非终结符的下一步状态，如规约，移进或 accept。

4.2.5 ParserTreeNode

该类是生成的语法分析树的节点，保存了 int id，boolean isLeaf 表示是否为叶节点，String symbol 表示对应 Token 的 type，String value 表示如果是叶节点则它的 lexValue 是多少，以及 List<ParserTreeNode> 表示它的儿子节点

4.2.6 ParserTree

该类保存了 ParserTree 的根节点

4.2.7 Parser

该类为语法分析器的核心，它使用了 Lexer 和 LRTable，parser 函数为解析函数，它每次读入 Lexer 的一个字符，同时维护符号栈和状态栈，根据 Token 的 type 查 LRTable 决定下一步动作：如果是移入则创建新的 ParserTreeNode，将 Token 中的 lexValue 放入，并令其为叶节点，压入符号栈; 如果规约则相应的将符号栈中的符号弹出与 Grammar 中的 Expression 比较能否成功规约，并创建新的 ParserTreeNode，其不是叶节点，并且儿子节点为符号栈中弹

出的可规约的 ParserTreeNode，规约完成后将这个节点压入符号栈，并根据 goto 表查找正确的状态名称压入状态栈。

这样在规约结束时即可获得一棵语法分析树。

5 实验结果

可执行代码见源码，使用方法见源码中的 README，源码可从 github 上查看下载，地址 https://github.com/zhang-x-z/CompilerPrincipal-Lab

测试选取了解析 JSON 格式的文件，并自己推到了 DFA 和 LR1 Table，词法包括 string，{，}，number，[，]，:，逗号，true，null，false 以及空格（包括制表符换行），输入的 dfa 文件如下

```
I0,0,{,I1,},I2,comma,I3,:,I4,[,I5,],I6,space,I7,cr,I7,lf,I7,
    tab,I7,t,I8,f,I9,n,I10,",I21,1,I24,2,I24,3,I24,4,I24,5,I24
    ,6,I24,7,I24,8,I24,9,I24,-,I25,0,I26
I1,1
I2,1
I3,1
I4,1
I5,1
I6,1
I7,1
I8,0,r,I11
I9,0,a,I12
I10,0,u,I13
I11,0,u,I14
I12,0,l,I15
I13,0,l,I16
I14,0,e,I17
I15,0,s,I18
I16,0,l,I19
I17,1
I18,0,e,I20
I19,1
```

```
I20,1
I21,0,^"\,I21,\,I22,",I23
I22,0,",I21,\,I21,/,I21,b,I21,f,I21,n,I21,r,I21,t,I21
I23,1
I24,1,0,I24,1,I24,2,I24,3,I24,4,I24,5,I24,6,I24,7,I24,8,I24
    ,9,I24,.,I27
I25,0,1,I24,2,I24,3,I24,4,I24,5,I24,6,I24,7,I24,8,I24,9,I24
    ,0,I26
I26,1,.,I27
I27,0,0,I28,1,I28,2,I28,3,I28,4,I28,5,I28,6,I28,7,I28,8,I28
    ,9,I28
I28,1,0,I28,1,I28,2,I28,3,I28,4,I28,5,I28,6,I28,7,I28,8,I28
    ,9,I28
```

lexer 配置文件如下

```
dfa.location=./simple-json-dfa.csv
dfa.startName=I0
dfa.encoding=utf8
sourcecode.encoding=utf8
sourcecode.buffersize=50
sourcecode.location=./example.json
dfa.I1=left-brace
dfa.I2=right-brace
dfa.I3=comma
dfa.I4=colon
dfa.I5=left-bracket
dfa.I6=right-bracket
dfa.I7=whitespace
dfa.I17=true
dfa.I19=false
dfa.I20=null
dfa.I23=string
dfa.I24=number
```

```
dfa.I26=number
dfa.I28=number
```

输入的语法文件如下

```
<?xml version="1.0" encoding="UTF-8" ?>
<grammar>
    <expression id="acc">
        <leftPart>
            S
        </leftPart>
        <rightPart>
            [object]
        </rightPart>
    </expression>
    <expression id="1">
        <leftPart>
            object
        </leftPart>
        <rightPart>
            [left-brace][right-brace]
        </rightPart>
    </expression>
    <expression id="2">
        <leftPart>
            object
        </leftPart>
        <rightPart>
            [left-brace][objects][right-brace]
        </rightPart>
    </expression>
    <expression id="3">
        <leftPart>
            objects
```

```
30          </leftPart>
31          <rightPart>
32              [string][colon][value][comma][objects]
33          </rightPart>
34      </expression>
35      <expression id="4">
36          <leftPart>
37              objects
38          </leftPart>
39          <rightPart>
40              [string][colon][value]
41          </rightPart>
42      </expression>
43      <expression id="5">
44          <leftPart>
45              value
46          </leftPart>
47          <rightPart>
48              [string]
49          </rightPart>
50      </expression>
51      <expression id="6">
52          <leftPart>
53              value
54          </leftPart>
55          <rightPart>
56              [number]
57          </rightPart>
58      </expression>
59      <expression id="7">
60          <leftPart>
61              value
62          </leftPart>
63          <rightPart>
```

```
64              [object]
65          </rightPart>
66      </expression>
67      <expression id="8">
68          <leftPart>
69              value
70          </leftPart>
71          <rightPart>
72              [array]
73          </rightPart>
74      </expression>
75      <expression id="9">
76          <leftPart>
77              value
78          </leftPart>
79          <rightPart>
80              [true]
81          </rightPart>
82      </expression>
83      <expression id="10">
84          <leftPart>
85              value
86          </leftPart>
87          <rightPart>
88              [false]
89          </rightPart>
90      </expression>
91      <expression id="11">
92          <leftPart>
93              value
94          </leftPart>
95          <rightPart>
96              [null]
97          </rightPart>
```

```
98      </expression>
99      <expression id="12">
100         <leftPart>
101             array
102         </leftPart>
103         <rightPart>
104             [left-bracket][right-bracket]
105         </rightPart>
106     </expression>
107     <expression id="13">
108         <leftPart>
109             array
110         </leftPart>
111         <rightPart>
112             [left-bracket][values][right-bracket]
113         </rightPart>
114     </expression>
115     <expression id="14">
116         <leftPart>
117             values
118         </leftPart>
119         <rightPart>
120             [value][comma][values]
121         </rightPart>
122     </expression>
123     <expression id="15">
124         <leftPart>
125             values
126         </leftPart>
127         <rightPart>
128             [value]
129         </rightPart>
130     </expression>
131 </grammar>
```

输入的 LR Table 文件如下：

1.LR Action Table

```
I0,left-brace,I2,0
I1,$,acc,2
I2,right-brace,I3,0,string,I4,0
I3,$,1,1
I4,colon,I6,0
I5,right-brace,I43,0
I6,string,I8,0,number,I9,0,true,I10,0,false,I11,0,null,I12,0,
    left-bracket,I15,0,left-brace,I16,0
I7,right-brace,4,1,comma,I17,0
I8,right-brace,5,1,comma,5,1
I9,right-brace,6,1,comma,6,1
I10,right-brace,9,1,comma,9,1
I11,right-brace,10,1,comma,10,1
I12,right-brace,11,1,comma,11,1
I13,right-brace,8,1,comma,8,1
I14,right-brace,7,1,comma,7,1
I15,right-bracket,I19,0,left-bracket,I20,0,left-brace,I30,0,
    string,I23,0,number,I24,0,true,I25,0,false,I26,0,null,I27
    ,0
I16,right-brace,I31,0,string,I4,0
I17,string,I4,0
I18,right-brace,3,1
I19,right-brace,12,1,comma,12,1
I20,right-bracket,I33,0,string,I23,0,number,I24,0,true,I25,0,
    false,I26,0,null,I27,0
I21,right-bracket,I35,0
I22,right-bracket,15,1,comma,I36,0
I23,right-bracket,5,1,comma,5,1
I24,right-bracket,6,1,comma,6,1
I25,right-bracket,9,1,comma,9,1
```

```
I26,right-bracket,10,1,comma,10,1
I27,right-bracket,11,1,comma,11,1
I28,right-bracket,8,1,comma,8,1
I29,right-bracket,7,1,comma,7,1
I30,right-bracket,I37,0,string,I4,0
I31,right-bracket,1,1,comma,1,1
I32,right-bracket,I39,0
I33,right-bracket,12,1,comma,12,1
I34,right-bracket,I40,0
I35,right-brace,13,1,comma,13,1
I36,left-bracket,I20,0,left-brace,I30,0,string,I23,0,number,
    I24,0,true,I25,0,false,I26,0,null,I27,0
I37,right-bracket,1,1,comma,1,1
I38,right-brace,I44,0
I39,right-brace,2,1,comma,2,1
I40,right-bracket,13,1,comma,13,1
I41,right-bracket,14,1
I42,right-bracket,15,1,comma,I36,0
I43,$,2,1
I44,right-bracket,2,1,comma,2,1
```

2.LR Goto Table

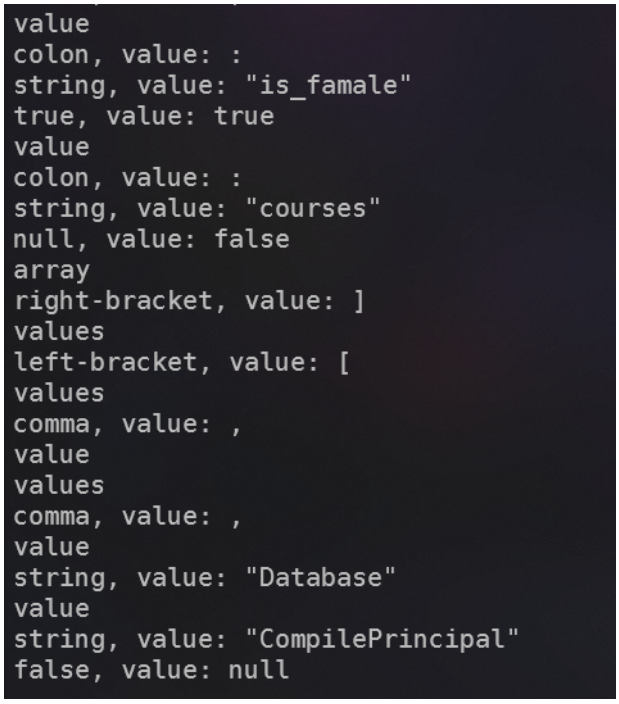
```
I0,object,I1
I2,objects,I5
I6,value,I7,array,I13,object,I14
I15,values,I21,value,I22,array,I28,object,I29
I16,objects,I32
I17,objects,I18
I20,values,I34,value,I22,array,I28,object,I29
I30,objects,I38
I36,values,I41,value,I42,array,I28,object,I29
```

parser 配置文件

```
lrTable.start=I0
lrActionTable.location=./json-actions.csv
lrGotoTable.location=./json-goto.csv
grammar.location=./json-grammar.xml
ignoreWhiteSpace=true
whiteSpace.name=whitespace
fileEncoding=UTF8
```

最后输出结果只是对语法分析树进行了层序遍历（顺序颠倒，输出的顺序是从右往左读每层语法分析树的结果），如图：

```
S
object
right-brace, value: }
objects
left-brace, value: {
objects
comma, value: ,
value
colon, value: :
string, value: "name"
objects
comma, value: ,
value
colon, value: :
string, value: "age"
string, value: "ZXZ"
objects
comma, value: ,
value
colon, value: :
string, value: "GPA"
number, value: 21
objects
comma, value: ,
value
colon, value: :
string, value: "is_male"
number, value: 3.92
objects
comma, value: ,
```



6 实验小结

通过本次实验，将词法分析与语法分析相结合，较为深入的理解了一个源文件是怎么样通过词法分析器变为一个 Token 流，然后再通过语法分析器产生一个 ParserTree 的过程，同时也通过这次实验对编译器有了一个总体上的认识，而不是停留在各个算法上，了解并亲自体会了总体的工作过程。

同时采用面向对象和软编码的思想，对 Lexer 和 Parser 进行一步步封装，最后一层 main 函数中只需要简单调用 parser 函数即可，体会到了封装的好处也提升了自己的编码能力。