

词法分析器实验报告

131130137—董本超



2016-10-20

NJU-SE

目录

# 1 实验目的

自己编写、调试一个词法分析程序，并对程序输入语句进行词法分析，从而更好的理解词法分析的原理。

# 2 内容描述

次程序用Java语言编写。程序读入一个内含程序语句的文本文件，对其中的内容进行分析，最终输出形式为<token类型, 识别的单词符号>的Token序列，token类型包括ID, KEYWORD, DIGIT, OPERATOR, ERROR。本词法分析器所分析的语言为C语言的缩减版，可识别C语言的保留字、变量名、操作符、数字等内容，但是不能处理注释、三目操作符（? :）、跳脱符（\）等，且不支持负的数值，引号中内容不可以跨行，所有支持的操作符和关键字将在下面表格中详细给出。

|  |  |  |  |
| --- | --- | --- | --- |
| 单词符号 | 种别码 | 单词符号 | 种别码 |
| int | 1 | + | 25 |
| unsigned | 2 | - | 26 |
| short | 3 | \* | 27 |
| char | 4 | / | 28 |
| long | 5 | & | 29 |
| float | 6 | && | 30 |
| double | 7 | | | 31 |
| if | 8 | || | 32 |
| else | 9 | ! | 33 |
| do | 10 | = | 34 |
| while | 11 | == | 35 |
| continue | 12 | += | 36 |
| for | 13 | -= | 37 |
| switch | 14 | \*= | 38 |
| case | 15 | /= | 39 |
| default | 16 | ++ | 40 |
| break | 17 | -- | 41 |
| struct | 18 | ; | 42 |
| typedef | 19 | ( | 43 |
| const | 20 | ) | 44 |
| static | 21 | [ | 45 |
| return | 22 | ] | 46 |
| id（变量名称） | 23 | { | 47 |
| digit（数值） | 24 | } | 48 |
| undefned | -1 | > | 49 |
|  |  | < | 50 |
|  |  | >= | 51 |
|  |  | <= | 52 |
|  |  | % | 53 |
|  |  | != | 54 |
|  |  | “ | 55 |
|  |  | ‘ | 56 |

表1 支持的Token表

# 3 方法思路

本次实验通过是基于最小或状态数以后的DFA来进行编程的，所以准备工作是要人工将以上定义的C语言缩减版转化成对应的NFA再转化成DFAo，然后程序通过DFAo并读入字符序列来判断并输出识别的Token序列，如果出现不识别的Token则将其归为ERROR类，即表1中种别码为-1的undefined。

# 4 最小化状态数的DFA

通过手工运算，得出本次实验所需的DFAo如下。

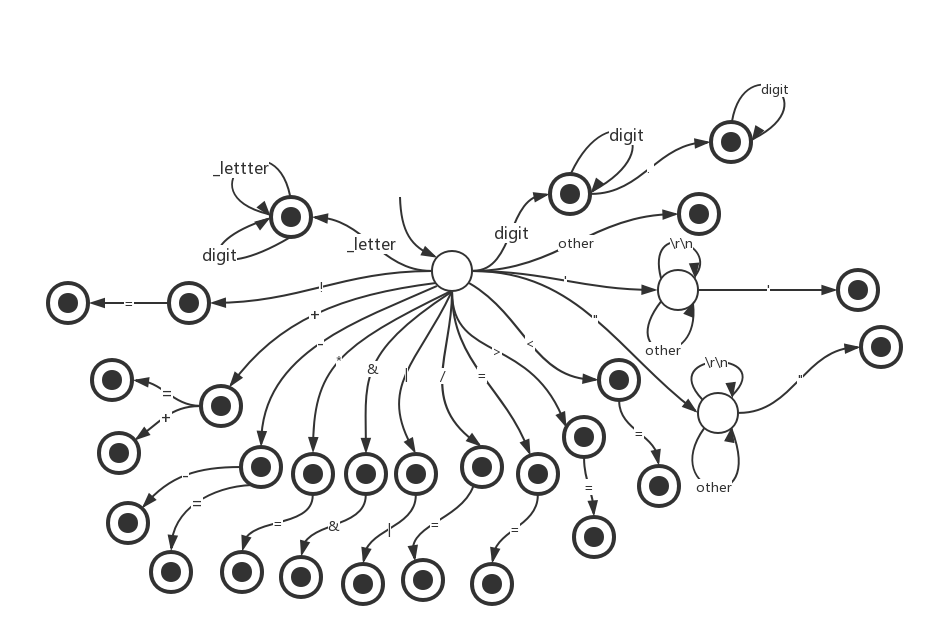
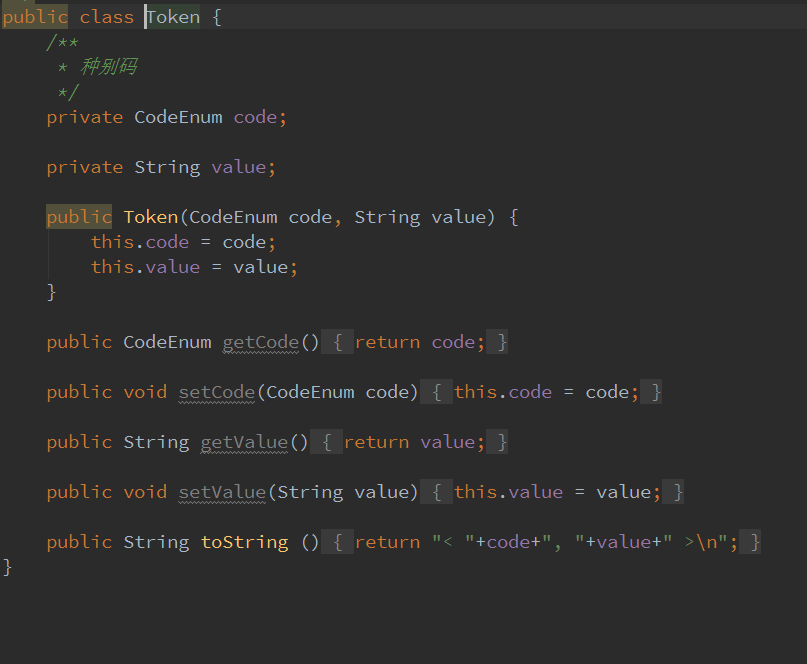


图1 DFA

上图中的由起点经过other路径到达的状态表示(, ), {, } 等类似的直接由起点到达终点的路径统称，因为他们几乎相同所以没有单独画出来。且每个状态都暗含了一个经过undefined到达ERROR状态的路径，图中也省略了。

# 5 主要数据结构

Token类是表示分析得到的词法单元，其中code可选值为ID, KEYWORD, DIGIT, OPERATOR, STRING, CHARACTER, ERROR，分别表示不同的类型。



# 6 核心算法

具体代码见Analyzer类的process()方法，该算法

# 7 运行截图

# 8 问题与解决方案

# 9 感受与总结