词法分析器描述文档

# 概述

本文档描述了一个词法分析器的设计及实现过程，实现了输入一段字符串代码经过分析后输出一个词法单元（Token）序列，一个常量符号表，一个变量符号表，以及输出词法错误位置的功能（当出现错误时输出错误行数）。（PPT给出的第一种实验方法）

## 功能需求

FR1 读入源程序代码

FR2 对源程序代码进行预处理，删除多余的空字符

FR3 使用DFA对源程序进行词法分析，生成词法单元序列，常量符号表，变量符号表

FR4 将生成的结果输出

FR5 报告分析过程中发现的错误的代码行数

# 主要实现步骤

## 设计正则表达式（RE）

1. 变量（id）

[a-zA-Z]([0-9]|[a-zA-Z])\*

1. 保留字

if

else

while

for

1. 数值常量

[1-9][0-9]\*.[0.9][0-9]\*

1. 符号

+ - \* / ; ( ) { }

< > <= >= == =

## 后续步骤

之后将RE转化为NFA，将NFA转化为DFA并最小化状态后，指导后续开发。

相关状态图如下

Letter -> [a-zA-Z]

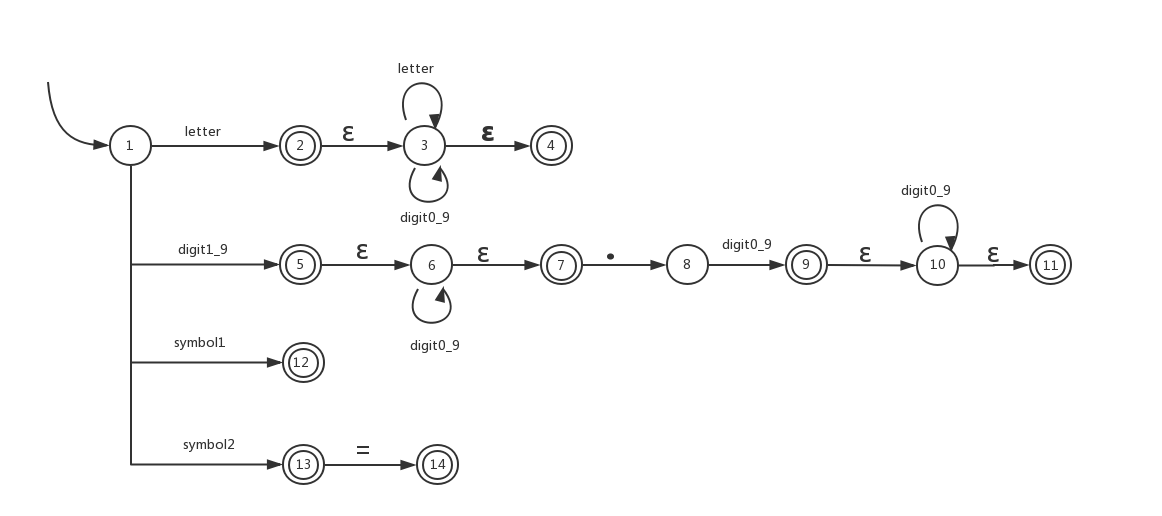
Digit0\_9 -> [0-9]

Digit1-9 -> [1-9]

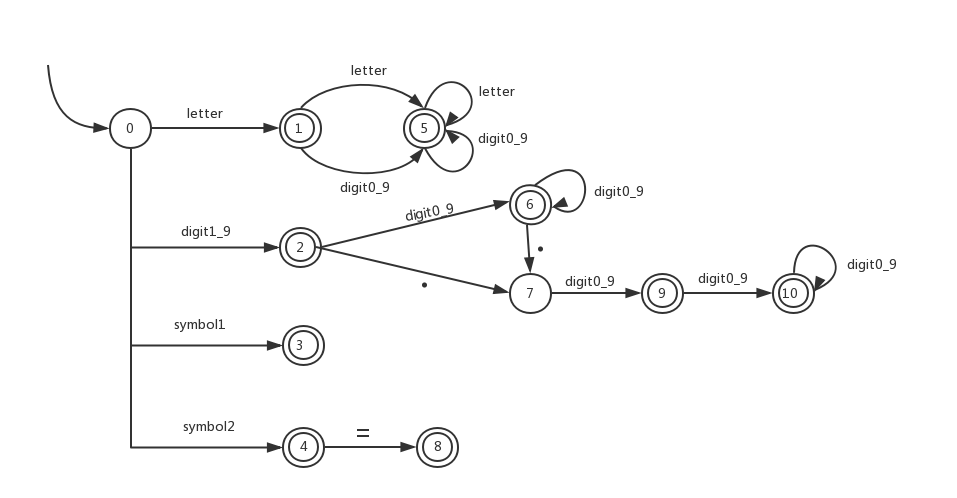
Symbol1 -> [+-\*/(){};]

Symbol2 -> [><!=]

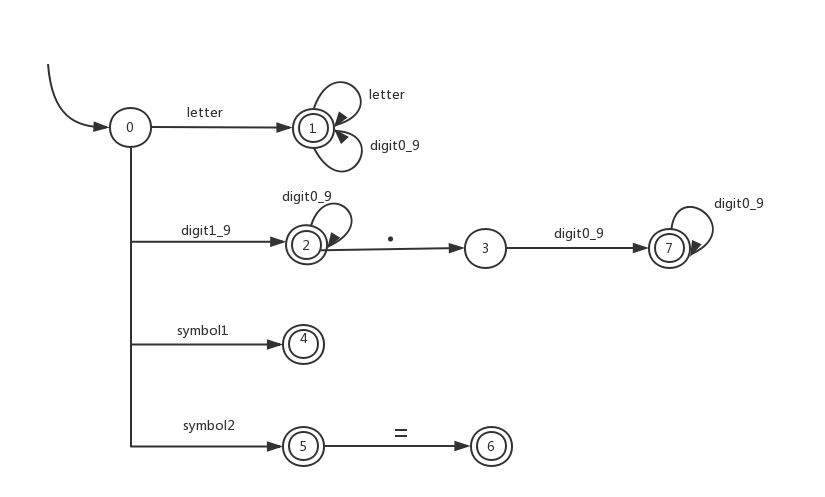
NFA



DFA



DFA\*



# 详细设计

## 各类职责

Io.FileHandler 负责文件读写

文件包括：

source.txt 待分析的代码

source\_path.config 配置source文件路径（在src下）

token.txt 输出token序列（出错时显示错误所在行数）

cons\_table.txt 常量符号表

var\_table.txt 变量符号表

main.Main 程序入口

analyzer.Scanner 负责扫描程序并删除多余空字符，将处理过的字符串传给Analyzer

analyzer. Analyzer 扫描字符串，并调用DFA 进行词法单元识别

analyzer.Table 定义了符号表的数据结构

analyzer.Token 定义了Token

analyzer.DFA.state.State 定义了状态接口，根据当前输入字符串决定下一个状态编号

analyzer.DFA.state.State0 – State7 实现了State接口

analyzer.DFA.DFA 有限自动状态机，持有所有状态对象，将读入的首个字母传给当前状态确定下一状态

## 主要算法

1. 将每一个状态独立成为一个类，每个状态有自己的标号，可根据传入的字符确定下一状态，知道自己是否是接受状态，如果是接受状态，可以返回识别的Token类型。如果需要增加状态这样可以容易修改。
2. 在 DFA中采用表驱动方式进行状态转换，DFA中持有一个List<State>，按照状态的标号依次存入List，将当前字符传入当前State后，可以获得下一个状态的标号，使用该标号可直接作为List<State>的索引找到下一个State。
3. DFA的接受状态

## 主要数据结构

DFA : List<State> 存储全部状态

Table(包括变量符号表和常量符号表) : List<Object>

输出形式： [VARIABLE,0,i] (type,index,id)

[CONSTANT,0,12.5] (type,index,value)

Token : 如果是保留字或符号 <reservedword>

如果是变量或常量 <type,index> index为当前变量(常量)在变量（常量）符号表中的index。

# 测试用例

输入：见根目录下source.txt

输出：见根目录下 token.txt , var\_table.txt , cons\_table.txt

其中，token.txt在输入的源代码没有错误时，保存token序列，否则保存错误行数提示。

Source.txt

for (i = 1;i < 10;i = i + 1);

sum = a + b

sum = c + d

var\_table.txt

[VARIABLE,0,i]

[VARIABLE,1,sum]

[VARIABLE,2,a]

[VARIABLE,3,b]

[VARIABLE,4,c]

[VARIABLE,5,d]

cons\_table.txt

[CONSTANT,0,1.0]

[CONSTANT,1,10.0]

token.txt

<FOR>

<L\_BRACKET>

<ID,0>

<ASSIGNMENT>

<CONSTANT,0>

<SEMICOLON>

<ID,0>

<SMALLER>

<CONSTANT,1>

<SEMICOLON>

<ID,0>

<ASSIGNMENT>

<ID,0>

<ADD>

<CONSTANT,0>

<R\_BRACKET>

<SEMICOLON>

<ID,1>

<ASSIGNMENT>

<ID,2>

<ADD>

<ID,3>

<ID,1>

<ASSIGNMENT>

<ID,4>

<ADD>

<ID,5>

# 错误处理

当出现错误时，分析终止，将发现错误的位置所在行数做为错误信息输出到token.txt。（仅输出程序出错的行数）

# 感受与总结

由于技术等原因，我选择了使用设计好的正则表达式实现词法分析器，RE -> NFA -> DFA -> 最少状态DFA的工作均手工完成，所以只要将课上有关词法分析部分的内容学懂后，基本没有太大的问题，编程难度相比第二种方法较小。但代码可读性较好，使用了面向对象编程，可修改性较好。虽然总体难度不大，但是由于要逐字处理字符串，以及指针的前后移动，很容易出现细节上的错误，在具体实现时需要十分仔细。