**词法分析器实验报告**

151250002卞纯源

目录

[1. 实验目的 1](#_Toc497682510)

[2. 内容描述 1](#_Toc497682511)

[3. 思路方法 2](#_Toc497682512)

[4.假设 2](#_Toc497682513)

[5.RE→DFA0过程 2](#_Toc497682514)

[5.1 RE 2](#_Toc497682515)

[5.2 RE->NFA 3](#_Toc497682516)

[5.3 NFA->DFA 3](#_Toc497682517)

[5.3 DFA->DFA0 3](#_Toc497682518)

[6.数据结构定义 3](#_Toc497682519)

[6.1 NFA的节点：Node 3](#_Toc497682520)

[6.2 NFA：NFA 3](#_Toc497682521)

[6.3 DFA／DFA0的节点：DFANode 4](#_Toc497682522)

[6.4 DFA / DFA0：DFA 4](#_Toc497682523)

[7. 核心算法 4](#_Toc497682524)

[8. 运行截图 9](#_Toc497682525)

[9.问题与解决 11](#_Toc497682526)

[10.感受与评价 12](#_Toc497682527)

# 实验目的

编写、调试一个词法分析程序，从正则表达式开始，构造NFA，构造DFA，最小化DFA，最后对语句进行词法分析，从而更好理解词法分析原理。

# 内容描述

此程序用java编写。程序读取一个文本文件，文件内容为事先定义好的正则表达式。然后程序按照RE->NFA->DFA-DFA0的顺序进行一步步的转换。转换完成后，再次读入一个文本文件，内容为一段java语言程序，并对其中的内容进行词法分析，此程序基本实现了对java程序的词法识别，可识别一部分保留字、标识符、操作符、部分标点符号，并输出格式为{“type=”,”code=”,”error=”}的Token序列，对不符合可识别格式的字符进行报错。

【注：此程序不识别空白符，如空格】

# 思路方法

1. 针对要识别的单词符号写出正则表达式
2. 程序自动构造出每个正则表达式对应的NFA
3. 使用子集构造法构造出DFA
4. 简化构造好的DFA，得到DFA0
5. 根据DFA0自动生成词法分析程序

5）代码思路参考核心算法部分

# 4.假设

AR1:保留字的优先级高于标识符

AR2:输入的文件是正常的java程序

# 5.RE→DFA0过程

## 5.1 RE

保留字：

(public)|(int)|(void)|(class)|(private)|(for)|(if)|(else)|(while)|(do)|(double)|(char)|(new)|(return)

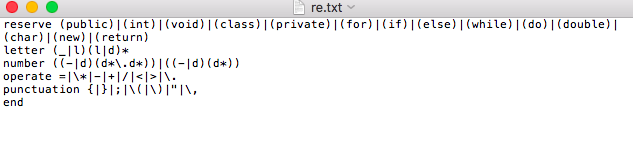
ID：

(\_|l)(l|d)\*

数字：((-|d)(d\*\.d\*))|((-|d)(d\*))

操作符：=|\\*|-|+|/|<|>|\.

标点：{|}|;|\(|\)|"|\,



## 5.2 RE->NFA

思路：首先用REHandler类里的reChange方法将RE改成后缀表达式。接着按字符读正则表达式，使用汤普森方法构造NFA。

数据结构：用NFA类表示构造的NFA，用Node类表示NFA的每一个节点。

## 5.3 NFA->DFA

思路：利用子集构造法，按NFA字母表中的字母查找当前节点的epsilon闭包和各条边的下一节点的闭包，构造新的DFA节点，从而构造DFA

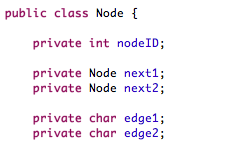
## 5.3 DFA->DFA0

思路：利用寻找等价DFA节点的方法，依次按边分开DFA状态，并回溯确认无可分状态，构造新的DFA0节点

# 6.数据结构定义

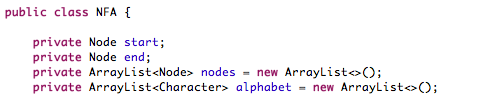
## 6.1 NFA的节点：Node

每个node最多有两条边通向下一节点，所以node类中包含了当前node的编号和下面节点的对应边和节点的指针



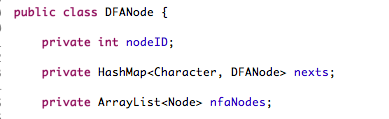
## 6.2 NFA：NFA

每一个NFA包括了开始节点和结束节点的指针，一个list存放所有的节点，一个list存放出现在正则表达式中的所有字符（即字母表）



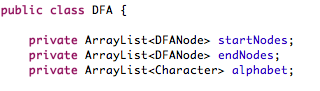
## 6.3 DFA／DFA0的节点：DFANode

每一个DFA节点可能用当前字母表数量个接邻节点，所以使用一个hashmap来存储当前节点和接邻节点的邻接关系。同时用一个list存储构成当前DFA节点的NFA／DFA节点的指针



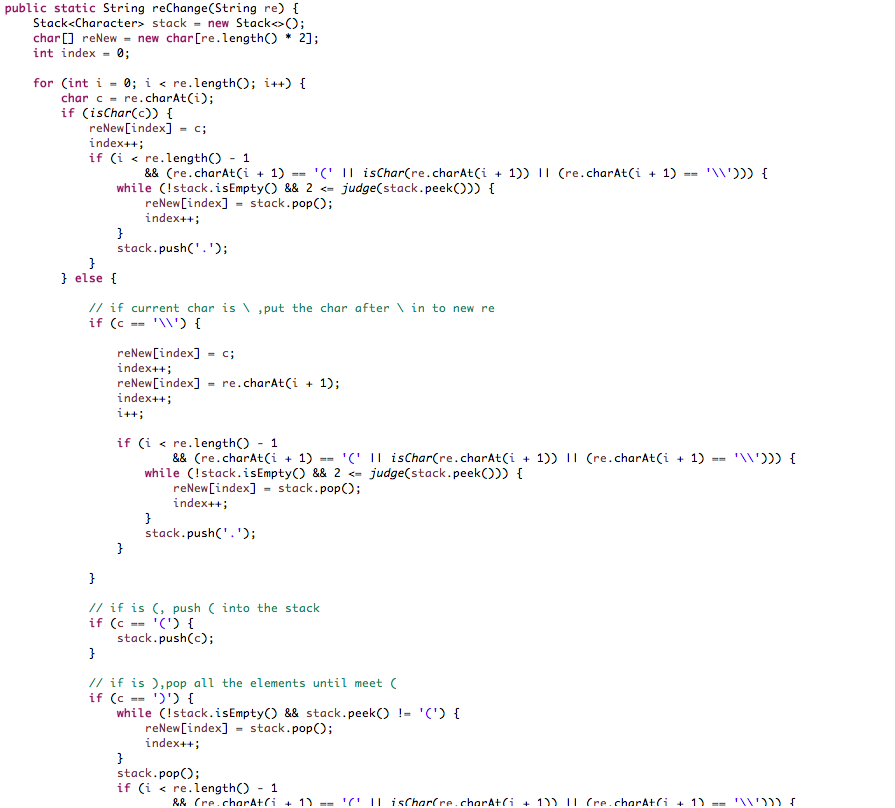
## 6.4 DFA / DFA0：DFA

DFA包括了开始状态的集合和结束状态的集合，同时用一个list存放字母表。

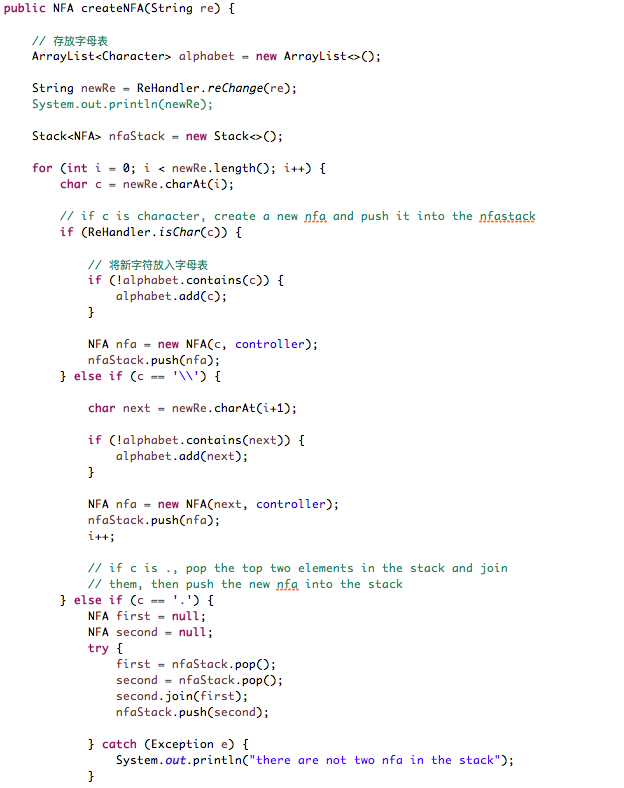


# 7. 核心算法

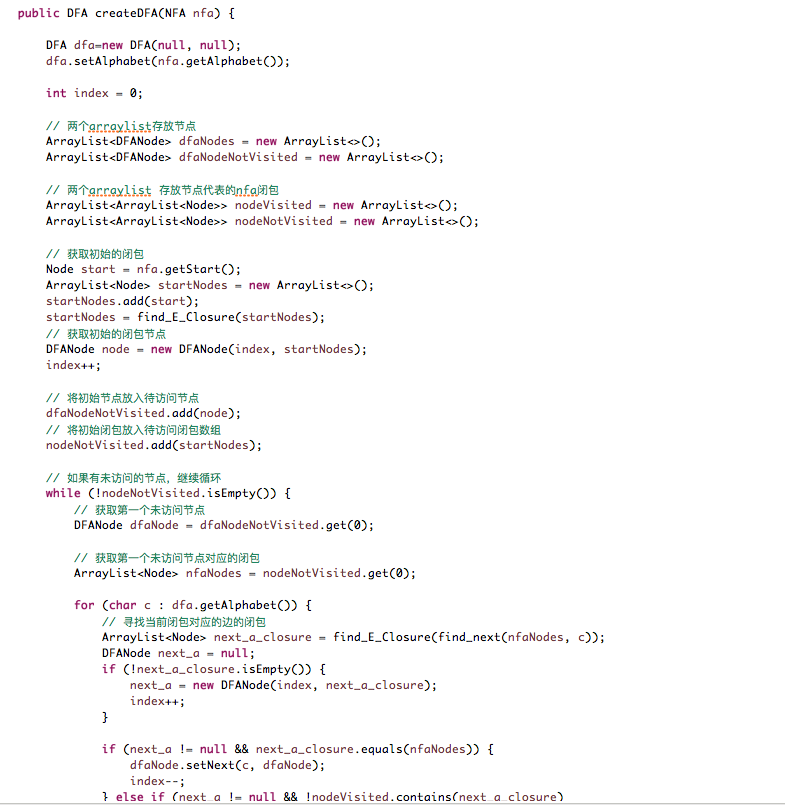
1、正则转后缀：如果当前字符是字母，直接输出并且判断是否需要添加连接符进入操作符栈，如果不是，按转义字符（\），括号（（）），连接符（.），重复（\*）进行相应操作



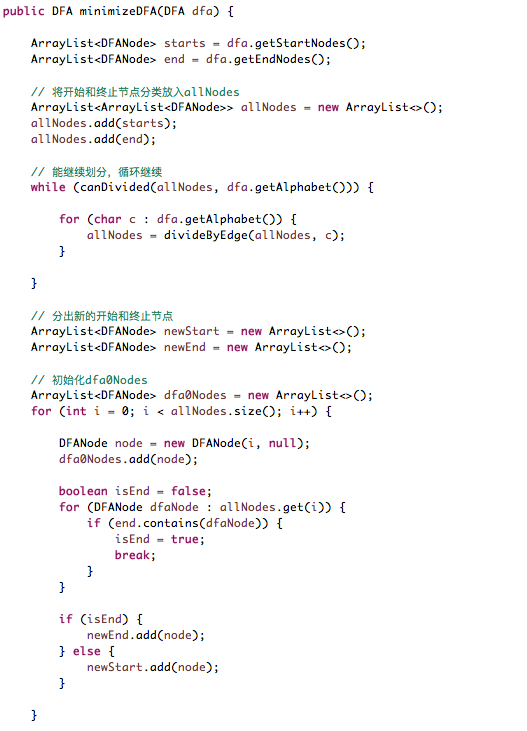
2、NFA构造：汤普森算法



3、NFA->DFA：子集构造法



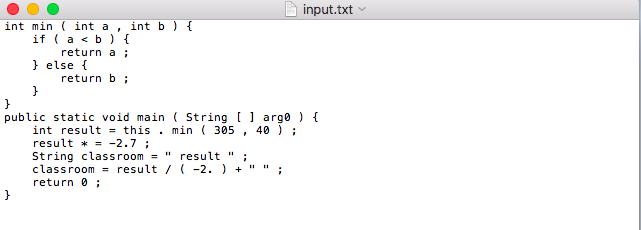
4、最小化DFA：寻找等价节点



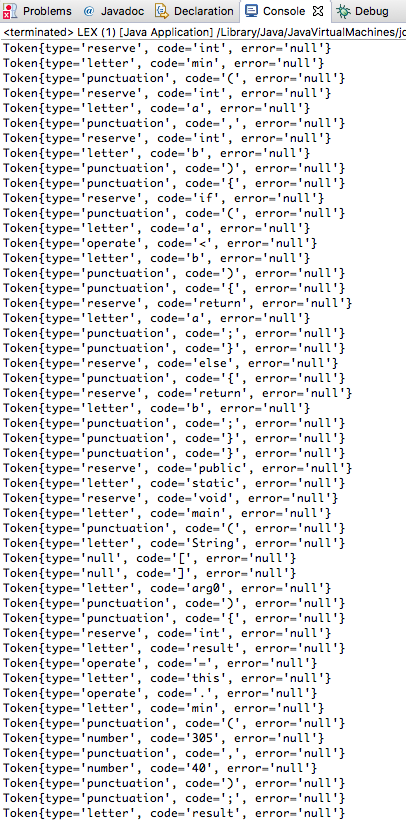
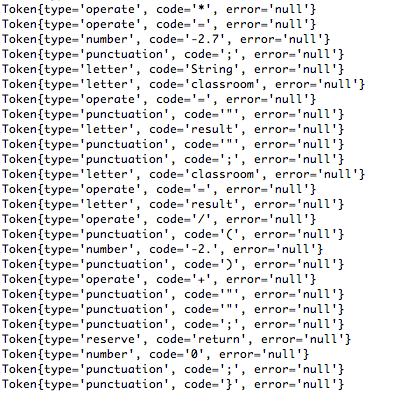


# 8. 运行截图

输入文件内容如下：



输出内容如下：

# 9.问题与解决

1）在进行关键字与ID标识符的判断时，采用的方法如下：

解决方案：一个正则表达式构造一个对应的DFA0，先对保留字的DFA0进行判断是否为保留字。

2）一开始只能判断正则表达式所包含的字母对应的字母串，不能识别一类字符串

解决方案：对读入的字符进行预处理，先将字符分类

3）一开始只考虑老师上课所讲的两个字符的NFA->DFA->DFA0，无法进行程序代码识别，只能识别两个字符组成的字符串

解决方案：在NFANode和DFANode中增加字母表，存放所有的边，每次寻找下一节点时遍历字母表

# 10.感受与评价

选择了第二种方案从RE开始构造词法分析器，刚开始的时候遇到了非常大的困难，因为对NFA和DFA的数据结构没有一个很好的认识，从而走了很多弯路。在构造DFA时，一开始只考虑了老师上课所讲的两个字符的情况，不能很好的扩展到多字符的情况。后来重新读龙书相应章节，对NFA和DFA的数据结构有了更深刻的认识和理解。在编写代码实现汤普森算法，子集构造法等算法的时候对其也有了更深刻的认识。这次的实验让我对老师上课讲的词法分析有了全新的认识，多了自己的理解。