**编译原理实验报告**

基于有限自动机的词法分析器构造

141250179 袁阳阳

目录

[1. 目标 1](#_Toc465265030)

[2. 内容概述 2](#_Toc465265031)

[3. 假设与依赖 2](#_Toc465265032)

[3.1实验环境 2](#_Toc465265033)

[3.2语言定义 2](#_Toc465265034)

[3.2.1保留字 2](#_Toc465265035)

[3.2.2符号 2](#_Toc465265036)

[3.2.3标识符 2](#_Toc465265037)

[3.2.4数字 2](#_Toc465265038)

[4. 主要思路和方法 3](#_Toc465265039)

[5. 相关的有限自动机描述 3](#_Toc465265040)

[5.1简单状态图 3](#_Toc465265041)

[5.2较复杂的状态图 4](#_Toc465265042)

[5.3数字 4](#_Toc465265043)

[5.4标识符 5](#_Toc465265044)

[5.5合并DFA 5](#_Toc465265045)

[6. 主要数据结构描述 6](#_Toc465265046)

[6.1 状态枚举类型 6](#_Toc465265047)

[6.2 token类型枚举类型 7](#_Toc465265048)

[6.3 token类 7](#_Toc465265049)

[6.4 保留字表 7](#_Toc465265050)

[7. 核心算法描述 8](#_Toc465265051)

[8. 测试用例 8](#_Toc465265052)

[9. 困难与解决方案 9](#_Toc465265053)

[9.1状态过多图形过于复杂问题 9](#_Toc465265054)

[9.2下一个字符位置确定和回退问题 9](#_Toc465265055)

[10.总结与收获 9](#_Toc465265056)

# 目标

本次实验的主要目的是对自定义的程序语言进行词法分析器程序构造，从而更好地理解词法分析原理。程序读取一个java程序文件，并对其中的内容进行词法分析，可识别保留字、变量名、数字、操作符（为简化程序选取一部分有代表性的操作符进行识别）、注释符（保留字优先于变量名），并输出格式为<tokenType, str>的token序列，并对未识别字符的异常情况进行报错处理。

# 内容概述

本报告主要描述了一个简单的词法分析器构造过程，包括实现过程中的理论推导，具体的核心算法和数据结构的描述，最终成品的功能概要和个人的感悟体会。

# 假设与依赖

## 3.1实验环境

操作系统：windows

编程语言：java

## 3.2语言定义

### 3.2.1保留字

可识别的保留字包括：class, public, protected, private, void, static, int, char, float, double, string, if, else, do, while, try, catch, switch, case, for

### 3.2.2符号

为了程序的简化，仅选取具有代表性的符号

选取由一个符号构成的有意义的运算符：+,-, /,\*,>,<,=;

由两个符号构成的有意义的符号：//

### 3.2.3标识符

由字母和数字组成（这里为了简化不包含下划线），开头必须为字母

### 3.2.4数字

仅考虑java范围内的整数，为简化不考虑浮点数和科学计数法。

# 主要思路和方法

1. 针对要识别的单词符号写出正则表达式
2. 构造出正则表达式对应的NFA
3. 合并所有NFA并简化为DFA
4. 基于DFA编写代码
5. 代码实现：根据当前所处状态，读取下一个字符，确定下一个状态，直到抵达终止状态，确定token的类型和内容。

对于上文列举和所要识别的符号和变量，其对应的Token type分别为：

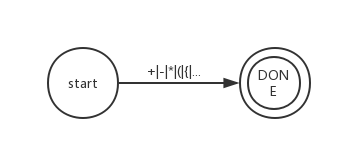
|  |  |
| --- | --- |
| 符号 | Token type枚举变量 |
| + | PLUS |
| - | MINUS |
| \* | TIMES |
| / | DIVIDE |
| > | GT |
| < | LT |
| = | ASSIGN |
| ( | LBRACKET |
| ) | RBRACKET |
| { | LBRACE |
| } | RBRACE |
| [ | LSBRACKET |
| ] | RSBRACKET |
| , | COMMA |
| ; | SEMICOLON |
| // | NOTE |
| 保留字 | RESERVED |

对于保留字的识别：因为保留字优先于标识符，所以在标识符状态下，每读取一个字符或者数字，都要与保留字表中的保留字一一比对，如果符合保留字，则退回多读取的符号，返回保留字的Token结构，如果都不符合，则正在处理的符号序列为标识符。

# 相关的有限自动机描述

## 5.1简单状态图

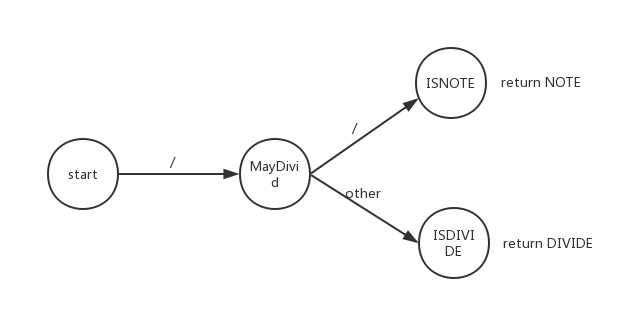
对于由一个符号组成的，无歧义的符号可由下面的自动机表示：



对于不能的符号，对应返回不同的Token种类。

## 5.2较复杂的状态图

具有多个选择条件但是无环且有限的情况，这里选取/和//为代表：



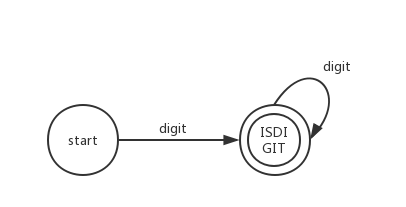
## 5.3数字

数字的正则表达式为：

num->digit+

digit->0|1|2|3|4|5|6|7|8|9

对应的DFA为：

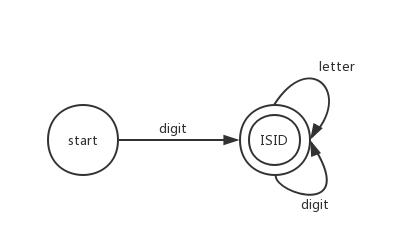


## 5.4标识符

标识符的正则表达式为：

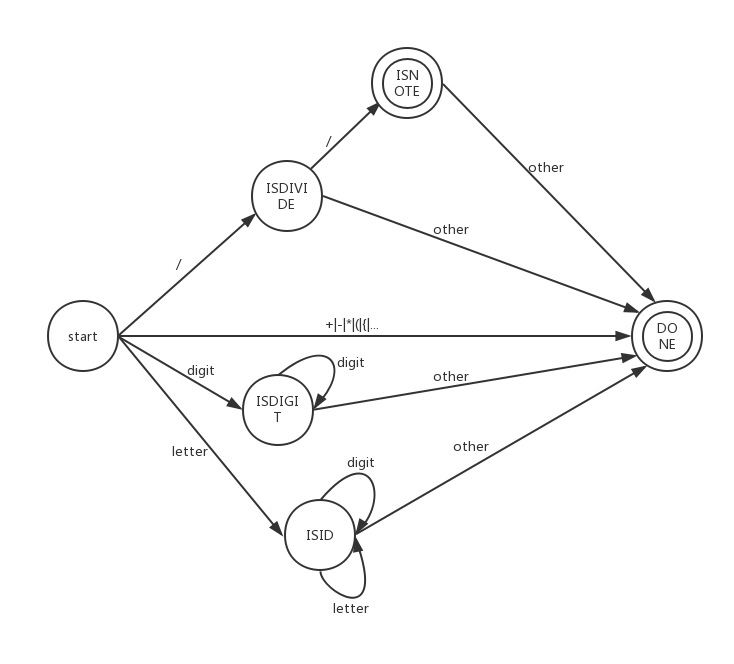
Id->letter|[letter|digit]\*

对应的DFA为：



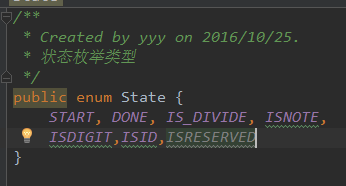
## 5.5合并DFA

合并后的DFA为：

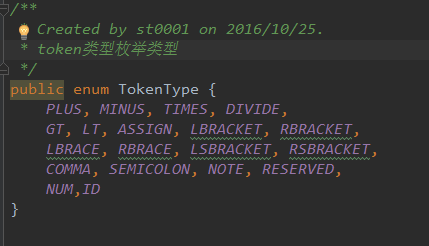


# 主要数据结构描述

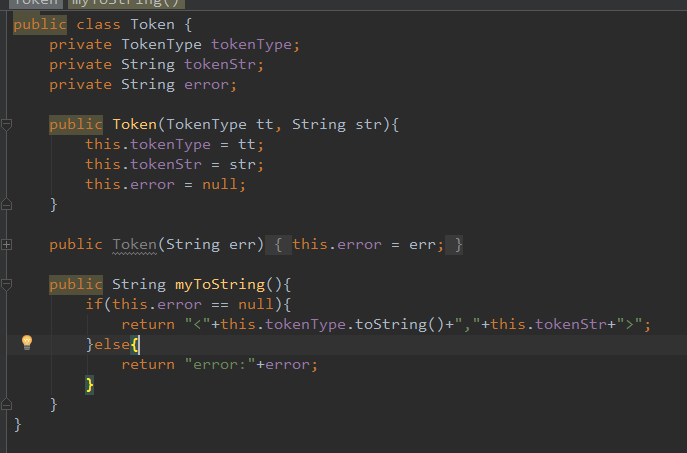
## 6.1 状态枚举类型



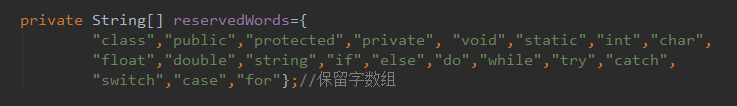
## token类型枚举类型



## token类



## 保留字表



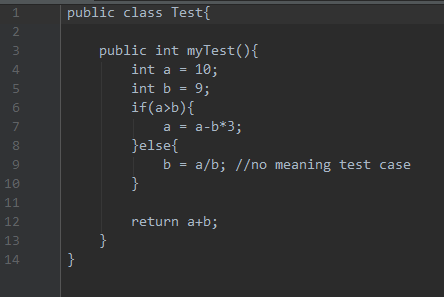
# 核心算法描述

本词法分析器的核心是scanner函数，函数接收一个输入的char数组，对数组中的字符进行遍历，通过switch判定当前所处状态，然后通过下一个字符判定下一个进入的状态。忽略空格和制表符，对于注释的内容不进行识别，对于无法识别的字符选择跳过，从下一个字符开始重新识别。

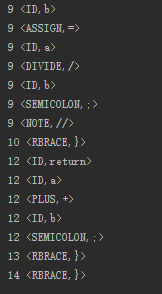
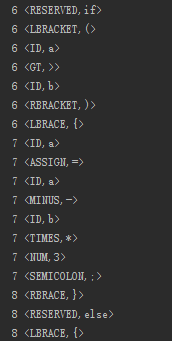
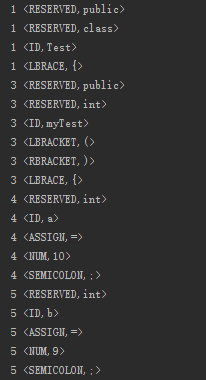
通过p变量判定当前识别的字符位置（前进、回退等操作），通过sp变量来存储被识别的字符。

# 测试用例

Input.txt



输出结果：



# 困难与解决方案

## 9.1状态过多图形过于复杂问题

在着手开始本次实验时，发现要考虑的情况很多，自动机构造很复杂。最终决定只选取必要的和具有代表性的符号和状态进行识别，减小了工作量但是内容兼具。

## 下一个字符位置确定和回退问题

由于单个符号，多个符号，整数和标识符的情况各异，原本想对指针做统一地前移，在需要的状态下进行回退，结果发现这样并不能减少工作量，反而因为状态过多，回退位置复杂而容易出现问题，最终确定针对不同的状态各自进行指针移动，并保证每识别完一个单词后，指针指向被识别单词的末尾。

# 10.总结与收获

通过本次实验，对于词法分析的原理和自动机的应用有了更好的理解和更深刻的认识，特别是对于将DFA转换为program这一步理解更为透彻，也收获了一点点小小的乐趣。