

语法分析实验报告

131130137—董本超



2016-11-3

NJU-SE

目录

[1 目标 2](#_Toc465959165)

[2 内容描述 2](#_Toc465959166)

[3 假设和依赖 2](#_Toc465959167)

[4 思路和方法 2](#_Toc465959168)

[5 主要数据结构描述 3](#_Toc465959169)

[5.1 状态栈和符号栈 3](#_Toc465959170)

[5.2 预测分析表 3](#_Toc465959171)

[6 核心算法 5](#_Toc465959172)

[7 测试用例 5](#_Toc465959173)

[8 困难与解决方案 7](#_Toc465959174)

[9 总结与收获 7](#_Toc465959175)

# 1 目标

自行定义文法， 运用语法分析方法对输入语句进行语法分析并输出结果， 加深对语法分析过程的理解。

# 2 内容描述

此次实验使用java编写。程序读取输入的token序列（如input.txt中所示），对其进行语法分析。这里使用LR(1)方法自底向上进行分析，最后输出归约的产生式序列。

定义的文法如下：

0 : S’ 🡪 S

1 : S 🡪 if C S else S

2 : S 🡪 id + id ;

3 : C 🡪 id > id

4 : C 🡪 C && C

# 3 假设和依赖

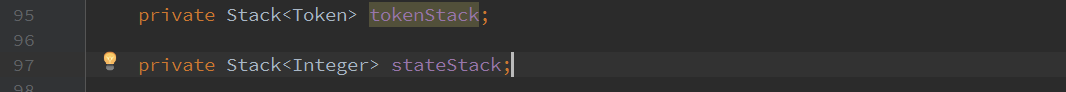
1. If后面必须有else，不允许单个if的情况
2. 所有的非终结符必须是单个字符

# 4 思路和方法

1. 构建符号栈，状态栈，建立 parsing table 在程序当中的映射，方便查询处理。
2. 读入用户输入的待分析的表达式， 依次移动读头， 如果根据当前栈顶的情况和读头下的符号来决定当前的动作。根据当前状态栈顶的状态号和读头下的符号查 parsing table，如果查到的是 Si 即 Shift 操作，则将读头下的符号压入符号栈，将 i 压入状态栈，移动读头到下一个字符；如果查到的是 ri 即 Reduce 操作，则将栈顶的所有与文法表达式 i右边相同的部分一次弹出， 同时将状态栈一起弹出， 将文法表达式 i 左边的符号 X 压入符号栈，查当前状态栈栈顶的项 I 在 GOTO 表当中的 GOTO(I,X)，将所得的状态号压入状态栈；如果读到 r0，则分析成功；如果有其他读入，则不合法。
3. 在读入过程中，如果出现查找不到 Action 或者 GOTO 表项，则报错，终止语法分析。

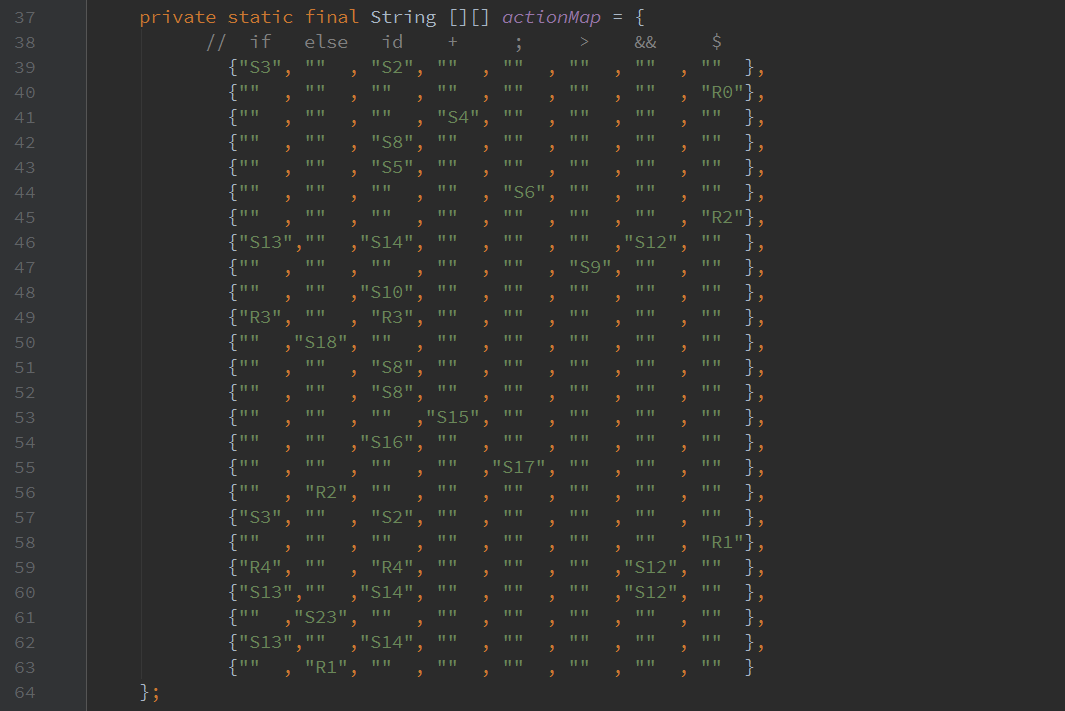
# 5 主要数据结构描述

## 5.1 状态栈和符号栈

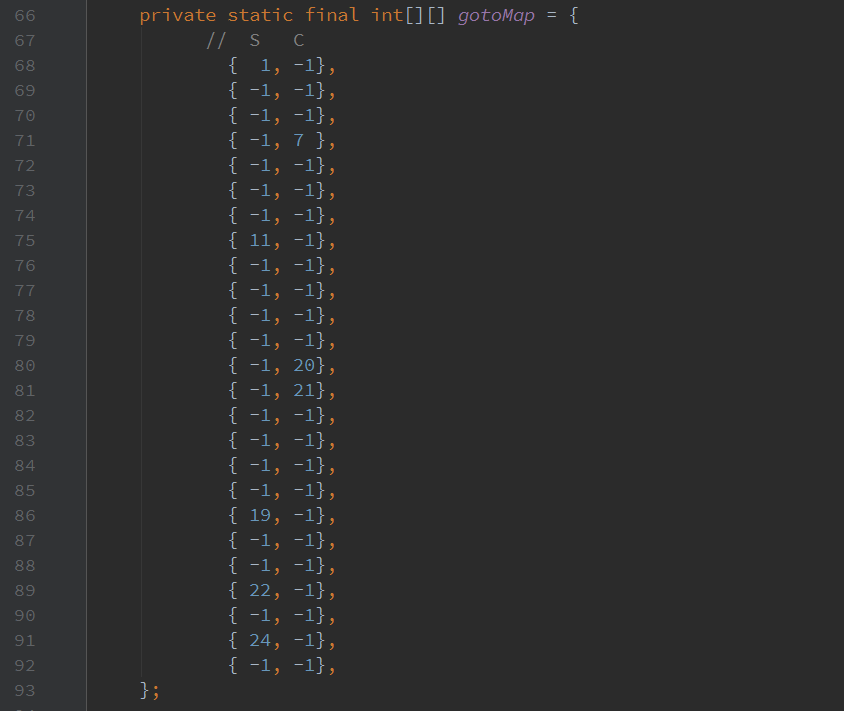


使用java.util.Stack<E>类库可以轻松构造状态栈和符号栈。

## 5.2 预测分析表



Action Map

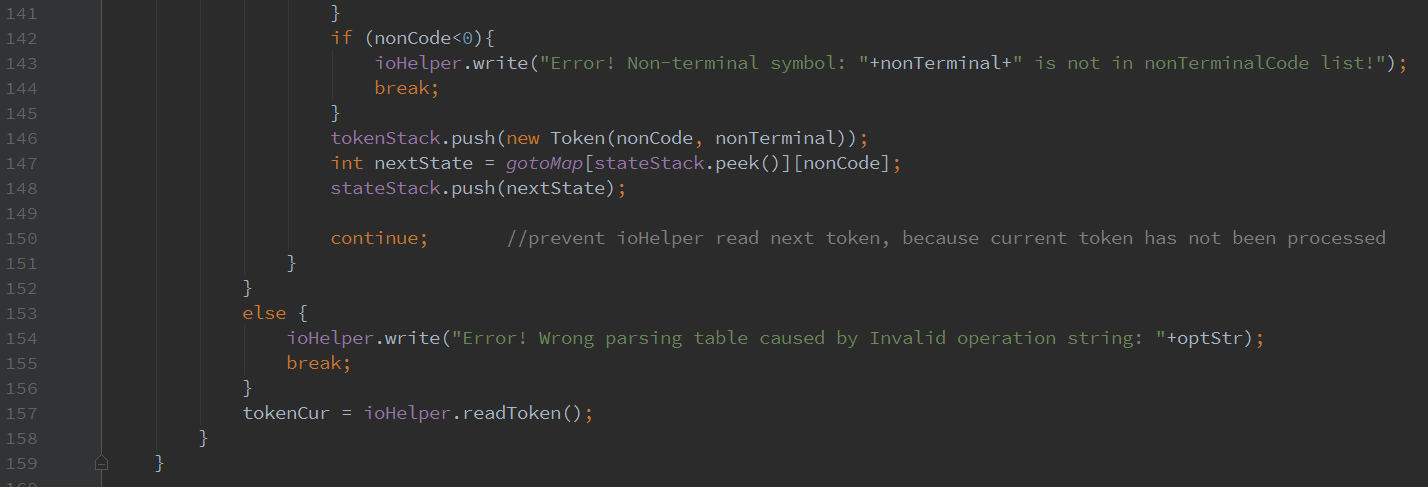


Goto Map

预测分析表使用二维数据存储，使用时只需将状态号和token的code编号带入数组下标即可获得action或者goto的操作。

# 6 核心算法

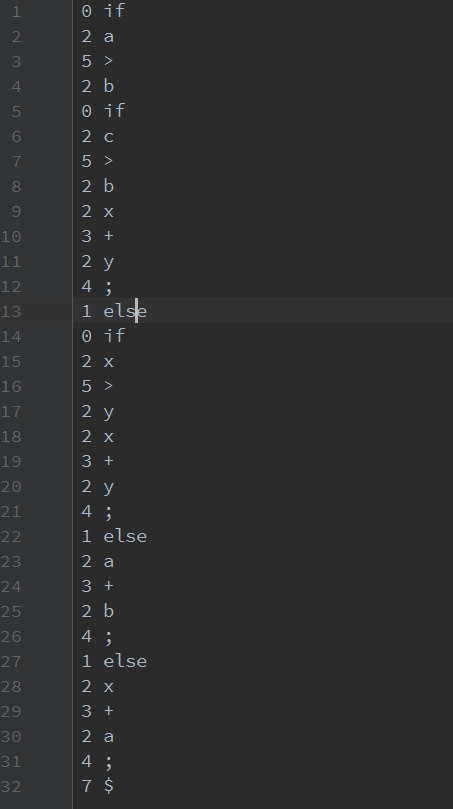




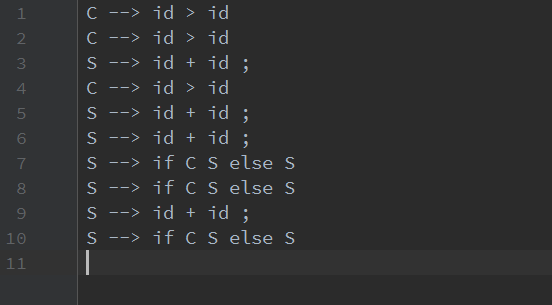
核心算法为parse() 方法，它从ioHelper获得一个一个的token，并根据状态栈中的状态代号和token中的code从actionMap中选择对应的操作，如果操作String是以‘S’开头的，则属于shift操作，如果以’R’开头，则是reduce操作，否则认为parsing table是非法的。

# 7 测试用例

输入token序列（code , value）：



输出归约产生式序列：



# 8 困难与解决方案

由于定义的文法需要手工画状态转换图，而问法又不是很简单，所以画的时候有25个状态，花了好长时间，中途还有画错的后来才纠正过来。

其他的并没有遇到什么大的困难。

# 9 总结与收获

加深了对LR(1)分析法的理解，更深刻的理解了语法分析器的工作过程。