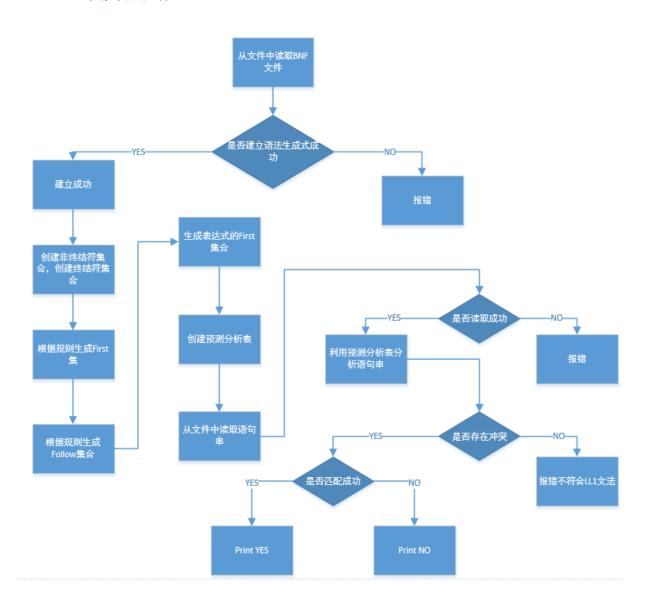
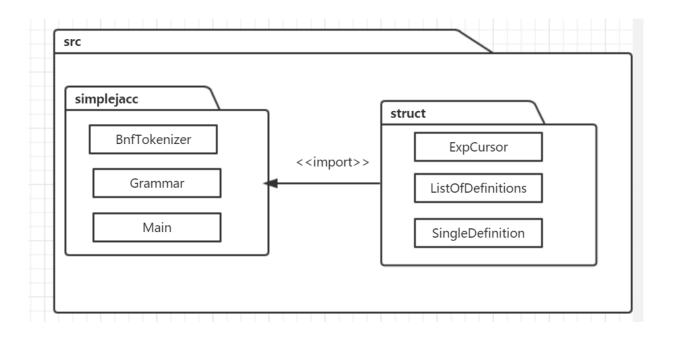
# SimpleJacc 文档说明

2014210173 陈世远

## 主要实现思路



相关包图、



### Main 函数(入口函数)

```
static String [] testFolder = "testcases/", "testcase2/", "testcase3/", "testcase6/", "tokenstream2.tok", "toke "tokenstream1.tok", "tokenstream2.tok", "toke "tokenstream3.tok", "tokenstr
```

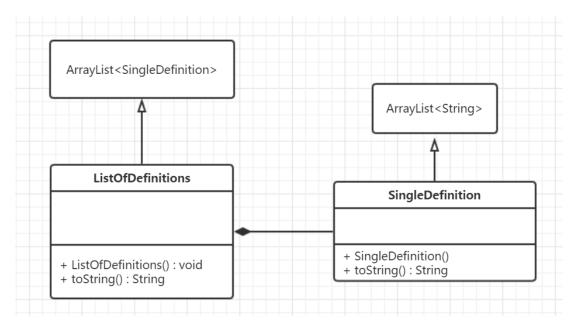
#### 类图(为了方便理解,自行在类图中添加了注释)

```
Grammar
- startNonTerminal : String
                                          //起始符号
- grammar : Map<String, ListOfDefinitions>
                                          //存储了生成式
nonTerminal : ArrayList < String >
                                          //非终结符号
- terminal : ArrayList < String >
                                         //终结符号
- firstSet : Map < String, Set < String > >
                                         //符号的First集合
followSet : Map < String, Set < String > >
                                        //非终结符号的follow集合
- definitionsFirstSet: Map<ExpCursor,Set<String>> //每个表达式的First集合
- analysisTables : ExpCursor[][]
                                             //预测分析表
+ Grammar(BufferedReader reader)
                                           //构造函数
+ addRule(String ruleText) : void
                                              //添加规则
- addToGrammar(String lhs, SingleDefinition singleDefinition): void //添加规则
- syntaxError(String rule): void
                                                     //语法错误
+ getDefinitions(String nonterminal): ListOfDefinitions //根据非终结符获取相关生成式
+ print(): void
                                                  //打印所有的语法规则
+ createNonterminal(): void
                                               //创建非终结符集合
                                                 //创建终结符号的集合
+ createTerminal(): void
+ createFirstSet(): void
                                               //创建FirstSet集合
+ getNonTerminalFirstSet(String nonterminal): void //获取非终结符号其First集合
+ createFollowSet(): void
                                                //创建FollowSet集合
+ getNonTerminalFollowSet(String nonTerminal): void //处理非终结符号的follow集
+ mylsNonterminal(String s) : boolean
                                                //通过检查是否在key的集合里来判断是否是非终结符号
+ createDefinitionsFirstSet(): void
                                                   //创建表达式的First
- getDefinitionFirstSet(SingleDefinition single, ExpCursor expCursor): void //创建每个表达式的First
+ createAnalysisTables(): void
                                                       //创建分析表
+ runGrammar(ArrayList<String> tok) : boolean
                                                      //利用预测分析表进行语法分析
+ readTok(BufferedReader reader):
                                                  //读取语句文件 , 检测是否合法
+ printArrayString(ArrayList < String > list) : void
+ printExp(ExpCursor expCursor) : void
+ getDefinitionByExpCursor(ExpCursor expCursor) : ArrayList<String>
+ buildGrammarAnalysisTables(): void
                                             //一口气创建
```

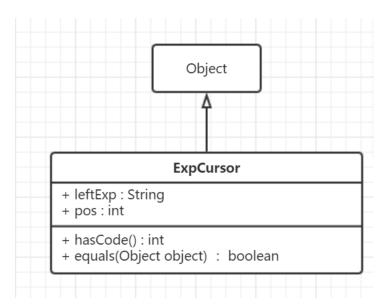
Grammar 类本身作为一个语法类,其中包括了所有的生成式,非终结符,终结符,预测分析表的数据成员。重要采用的原始数据结构主要为 Map 和 ArraryList。其中最为重要的产生式的存储使用的数据结构为 Map<String,ListOfDefinitions>

并提供了 readTok(BufferedReader reader)方法和 runGrammar(ArrayList<String> tok)方法来分析输入的语句是否符合这个语法。由于时间的问题,这个类最重要,但是写的并不是很好。

在测试过程中发现一点,如果存在左递归的情况。会崩溃。因为 first 集和 follow 集一直递归直到栈溢出。

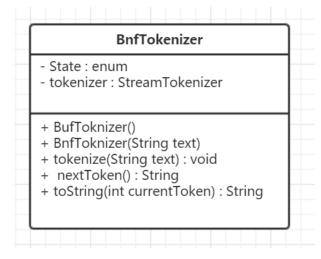


ListOfDefinitions 类和 SingleDefinition 他们均继承于 ArrayList。是语法规则产生式的结构元素。



其实写这个类的主要目的是作为在 Map<String,ListOfDefinitions>中定位产生式的一个结构。

而重写了 hasCode()和 equals(Object object)是为了可以在 Map 的数据结构中充当起 Key



这个是作为从数据流中解析并读取 BNF 范式的词法分析类。

## 运行结果

```
| Campus |
```

# 总结和思考

本来打算好好的使用面向对象的思想去完成,可是到后期时间各方面的安排原因,导致没有很好的完成。

整个完成的时间,大概有一半的时间在调试的过程中。对编译原理课程的知识如果没有比较熟悉的了解。会花很多的时间在调试。

但是在实践的过程中也加深了对一些知识点的理解。亲手测试,会有不一样的体会。

我想以后有时间的话,说不定会去认真完成一下。因为我在查找资料的时候,发现网络上用 Java 写的 LL1 语法分析器的例子很少,十分想写好一点发布到 Github 上去。