# 编译原理 PJ 报告——MiniJava 编译器

14307130205 魏君毅

## 一、工具

该项目独立完成，使用 Java1.8 作为编程语言。

Github 地址为：https://github.com/cjwjy007/MiniJavaCompiler

词法语法生成工具有很多，如 Flex/Bison，Lex/Yacc，Jlex/CUP，JavaCC, ANTLR。

我选择的是 ANTLR，原因是：该工具有详细的文档和说明，上手容易；提供了便利的词语语法生成，仅需一个文法文件便可以输出词法语法分析类，自动生成语法树；提供了错误处理的简单功能，提供了接口供用户自定义处理错误；该工具有 IDEA 支持的插件，便于调试。

## 二、ANTLR 文法文件

ANTLR4 的文法文件后缀为.g4。在使用该工具时，第一个步骤便是编写文法文件，文法文件要编写两个部分，parser rules 与 lexer rules，即语法分析与词法分析。注意到词法分析以大写字母开头，语法分析以小写字母开头。文发文件的开头先声明语法为 MiniJava，给出类的入口函数，之后编写语法与词法规则。这些规则在 MiniJava 的 BNF 中有详细的介绍。

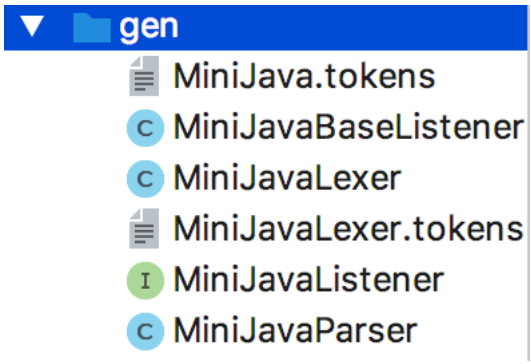
由于涉及到自定义的错误处理，我在语法分析中添加了几种对错误情况的匹配，若遇到这些错误情况，将会通知错误监听器并初始化错误处理的类。

下面两张图片分别为语法与词法分析的规则，来自 MiniJava.g4 文件。

```
//parser rules
mainClass: 'class' ID '{' 'public' 'static' 'void' 'main' '(' 'String' '[' ']' ID ')' '{' statement '}' '}';
classDeclaration : 'class' ID ('extends' ID)? '{' (varDeclaration)* (methodDeclaration)* '}';
varDeclaration   : type ID ';';
methodDeclaration: 'public' type ID '(' (type ID (',' type ID)*)? ')' '{' (varDeclaration)* (statement)* 'return' expr ';' '}';
type    : 'int' '[' ']'
        | 'boolean'
        | 'int'
        | ID
        ;
statement: '{' (statement)* '}'
         | 'if' '(' expr ')' statement 'else' statement
         | 'while' '(' expr ')' statement
         | 'System.out.println' '(' expr ')' ';'
         | ID '=' expr ';'
         | ID '[' expr ']' '=' expr ';'
         ;
expr    : expr ('&&'|'<'|'+'|'-'|'*') expr
        | expr ('&&' | '<' | '+' | '-' | '*' )
               {notifyErrorListeners(this.getCurrentToken(),
                   "Missing right operand",
                   new MyRecognitionException(this));}
        | ('&&' | '<' | '+' | '-' | '*' ) expr
               {notifyErrorListeners(this.getCurrentToken(),
                   "Missing left operand",
                   new MyRecognitionException(this));}
        | expr '[' expr ']'
        | expr '.' 'length'
        | expr '.' ID '(' (expr (',' expr)*)? ')'
        | expr '.' ID '(' (expr (',' expr)* )? ')' ')'
                     {notifyErrorListeners(this.getCurrentToken(),
                         "Too many parentheses",
                         new MyRecognitionException(this));}
        | expr '.' ID '(' (expr (',' expr)* )?
            {notifyErrorListeners(this.getCurrentToken(),
                "Missing closing ')'",
                new MyRecognitionException(this));}
        | INT_LITE
        | 'true'
        | 'false'
        | ID
        | 'this'
        | 'new' 'int' '[' expr ']'
        | 'new' ID '(' ')'
        | '!' expr
        | '(' expr ')'
```

```
//lexer rules
ID :  [a-zA-Z_$][a-zA-Z0-9_]*;
INT_LITE : ('0' | [1-9] [0-9]*);
WS      : [ \t\r\n]+ -> skip ;
BLOCK_COMMENT : '/*' .*? '*/' -> skip;
LINE_COMMENT: '//' ~[\r\n]* -> skip;
ASSIGN  : '=';
GT      : '>';
LT      : '<';
GE      : '>=';
LE      : '<=';
PLUS    : '+';
MINUS   : '-';
TIMES   : '*';
BANG    : '!';
AND     : '&&';
OR      : '||';
L_PAREN : '(';
R_PAREN : ')';
L_BRACK : '[';
R_BRACK : ']';
L_BRACE : '{';
R_BRACE : '}';
COMMA   : ',';
DOT     : '.';
SEMI    : ';';
```

通过 ANTLR 的 generator，我们可以生成词法与语法的分析类，如下图：




图中，MiniJavaLexer.java，MiniJavaParser.java 则是对应的词法分析器和语法分析器，MiniJavaListener.java 实现了一个访问语法树的接口，MiniJavaBaseLisener.java 实现了这个接口。关于错误分析，若要自定义错误，我们应该继承 BaseErrorListener 类，在 Main 函数中重写监听器。

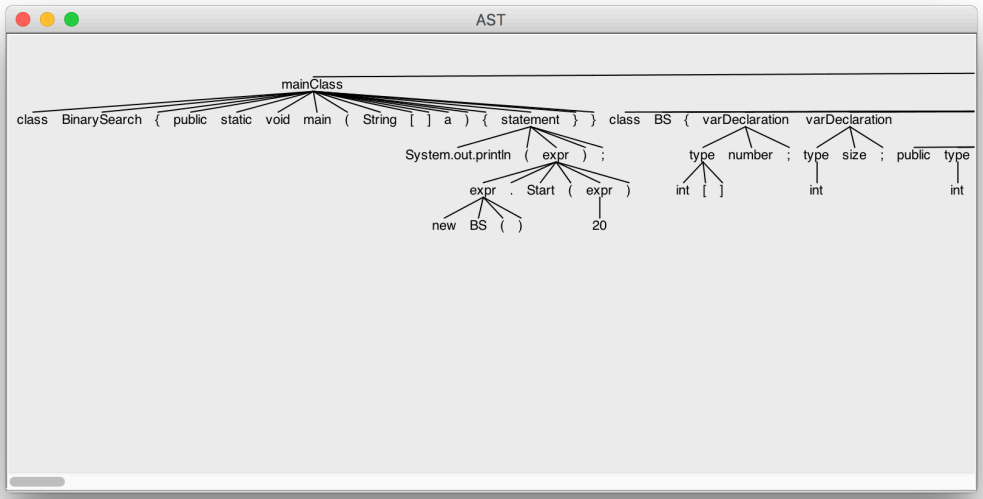
## 三、调用 ANTLR

```
public class Main {
    public static void main(String[] args) throws IOException {
        //get input
        String inputFile = "test/binarysearch.java";
        CharStream input = CharStreams.fromFileName(inputFile);

        //lexical analysis
        MiniJavaLexer lexer = new MiniJavaLexer(input);
        CommonTokenStream tokens = new CommonTokenStream(lexer);

        //syntax analysis
        MiniJavaParser parser = new MiniJavaParser(tokens);
        parser.removeErrorListeners();
        parser.addErrorListener(new ErrorReportListener());
        ParseTree tree = parser.init();

        // Show AST in GUI
        JFrame frame = new JFrame( title: "ANTLR AST");
        JPanel panel = new JPanel();
        TreeViewer viewer = new TreeViewer(Arrays.asList(parser.getRuleNames()), tree);
        panel.add(viewer);
        JScrollPane scrollPanel = new JScrollPane(panel, JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
                JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);
        frame.getContentPane().add(scrollPanel);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setSize( width: 800,  height: 400);
        frame.setVisible(true);
    }
}
```

Main 函数如上图所示，首先，先读入从 MiniJava 官方网站下载

的示例 binarysearch.java 文件，之后调用词法分析器找到 minijava 文件中的 tokens，然后调用语法分析器生成一课抽象语法树，在这之前，我们要先移除它原有的错误监听器，加上我们自定义的监听器，之后调用 parser.xx()，这个 xx 即为文法文件中的入口函数，得到抽象语法树。

若文件正确，我们则通过 GUI 显示这课抽象语法树，调用 swing 控件，将 treeview 显示在上边，最后结果如下图所示。




## 四、错误处理

ANTLR自动生成的BaseErrorListener为我们处理了一些错误的监听，我们将继承这个类。创建MiniJavaBaseErrorListener，重写句法错误的函数syntaxError，在该函数中输出我们自己的错误报告与下划箭头报告。

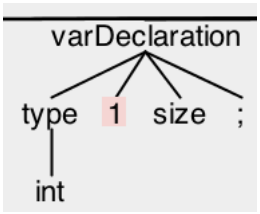
处理的错误类型包括：

1. 词法错误——变量声明错误。

如变量声明开头包含数字，定义int 1size，错误信息如下图：

```
line 12:8 at [@38,272:272='1',<21>,12:8]: extraneous input '1' expecting ID
    int 1size;
        ^
```

此时输出的抽象语法树：



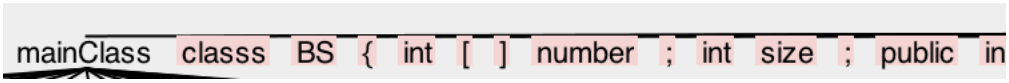

可以看到，此时不仅给出了错误信息，还给出了错误的原因，并将错误的地方用红色的下划箭头标出。之后的所有错误处理都讲与之类似。

2. 词法错误——关键词错误

如关键词输入有误，将class改成classs，错误信息如下图：

```
[Lexical Error]: line 10:0  Wrong Keyword: classs   Expected Tokens: {<EOF>, 'class'}   Suggested Keyword: class
classs BS{
^^^^^^
```

此时输出的抽象语法树：

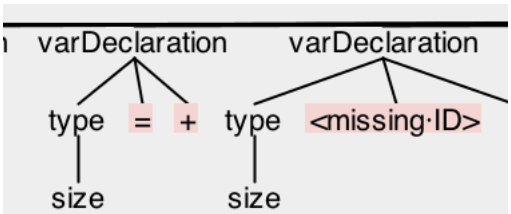



3. 语法错误——操作数缺失

该错误为运算符左右操作数的缺失，如size = + size，错误如下：

```
[Syntax Error]: line 13:6   Misuse Reserved Word: = Should be: ID
    size = + size;
      ^
line 13:14 at [@44,292:292=';',<44>,13:14]: missing ID at ';'
    size = + size;
             ^
```

此时输出的抽象语法树：

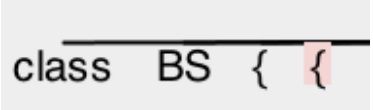


## 4. 语法错误——括号不匹配

该错误为括号不匹配，类声明后多输入一个{，错误如下：

```
line 10:9 at [@32,244:244='{',<40>,10:9]: extraneous input '{' expecting {'public', 'int', 'boolean', ID, '}'}
class BS{{
         ^
```

此时的抽象语法树：

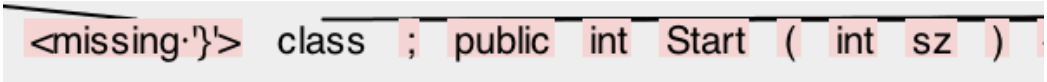



## 5. 语法错误——定义变量错误

若将变量定义为关键词，如int class，错误如下：

```
[Syntax Error]: line 13:5   Misuse Reserved Word: class Should be: ID
    int class;
        ^^^^^
```

此时的抽象语法树：

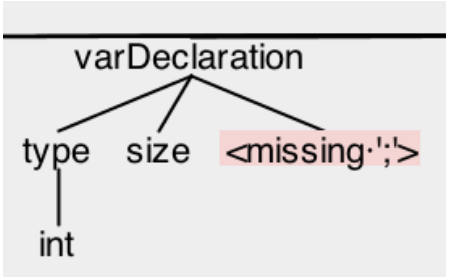



## 6. 语法错误——分号缺失

输入int size int size1;错误如下：

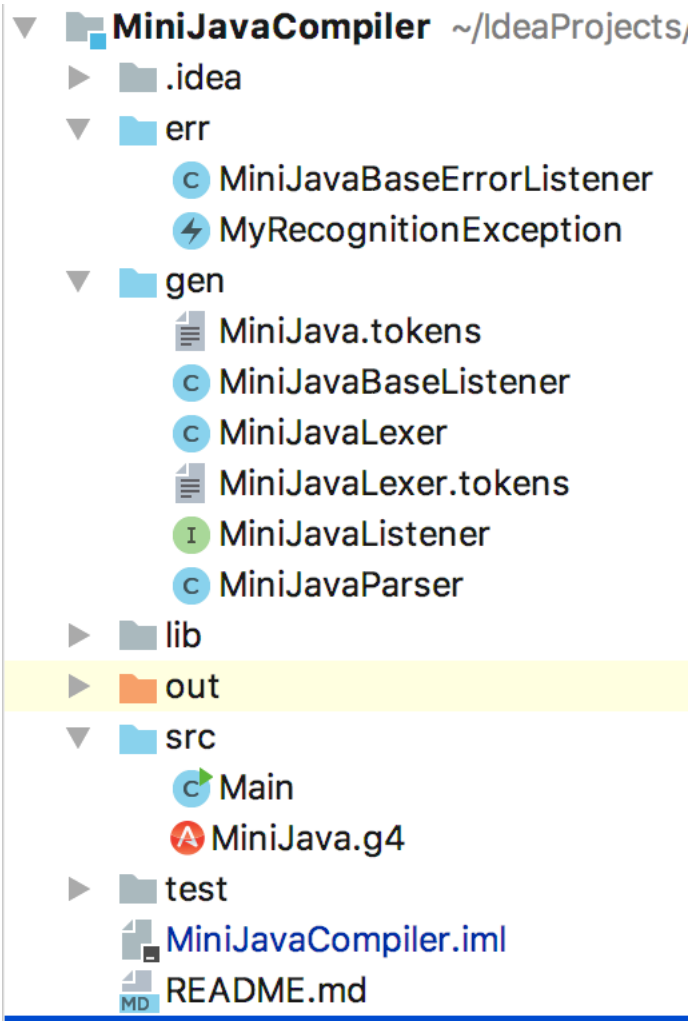
```
line 12:13 at [@39,277:279='int',<9>,12:13]: missing ';' at 'int'
    int size int size1;
             ^^^
```

此时抽象语法树：



## 五、项目结构

项目结构如下所示，src为目录下为Main函数类与文法文件，gen下为ANTLR4自动生成的类，err下为错误处理类，test中存放minijava的测试代码，lib中存放ANTLR4的JAVA包。




## 六、感想

该项目使我了解了编译器是如何通过词法和语法分析一步步将我们编写的代码变成一颗电脑可以识别的抽象语法树。对错误处理的重新编写也让我对Java中几种编译的常见错误进一步加深了印象。

将来有时间可以尝试对编译器后端的开发，使自己更加了解编译的知识和原理。