

# 编译 Project 报告

顾瑞珏 14300270021 乔萃雯 15307130322

Repo 地址: <https://github.com/jelly62/minijava-compiler>

## 一、工具使用

### • 工具对比

目前，有如下的一些主流的词法/语法分析器：

#### (1) Lex/Yacc

生于 Unix，是最经典的词法/语法分析器，是经典教材中的示例御用工具。现在它也支持在 Windows 上生成（安装环境），然而其所生成语法分析器的语言仅有 C 语言。

#### (2) Flex/Bison

与前者类似，Bison 与 Yacc 有很高的兼容性。生成语言为 C、C++ 和 Java。

#### (3) CoCo/R

较早的一个语法分析器生成工具。其生成语法分析器的语言极其之多，包括 C#、Java、C++、F#、VB.Net、Oberon 等等。

#### (4) ANTLR

作为翻译程序的一部分，你可以使用简单的操作符和动作来参数化你的文法，使之告诉 ANTLR 怎样去创建抽象语法树 (AST) 和怎样产生输出。ANTLR 知道怎样去生成识别程序，生成语法分析器的语言包括 Java, C++, C#。

### • 选择工具原因

我们选择的工具需要符合以下几条标准：

支持 Windows 系统下的操作；

支持 Java 语言的编译；

操作、安装简易，易懂；

支持树的生成。

综上，我们挑选了 Antlr4 作为我们的编译器。

## 二、词法、句法分析原理

## • 分析的主要顺序

g4 文法文件通过 antlr4 工具，生成 lexer 词法解析器和 parser 语法解析器以及 visitor 和 listener 的 java 文件或其他语言文件（支持多语言）。

## • 各部分原理

### Lexer:

对语句进行词法分析，把输入切分为不同的 token，可以对语句中的重要信息进行处理。

词法根据输入的字符流，识别出符合语言定义的单词，这样我们就能够得到一个单词流，而非字符流。

### Parser:

根据输入的单词流，识别出其中的语法结构。parser 用作句法分析，是字符串和 lexer 的组合，用来匹配分析一个句子。

### g4 Rules(parser 和 lexer 规则):

以:开头，以;结尾。多行规则以“|”竖线符号分隔;

lexer 定义时名字以大写字母开头。parser 定义时名字以小写字母开头。

## 三、代码结构及原理

在我们的 repo 中，我们的源代码结构主要分为：

### 1. Minijava.g4

在 Minijava.g4 中，我们定义了整个 Minijava 的语法，包括错误处理机制。Antlr4 会将 minijava 定义的 grammar 编译，生成 lexer 和 parser。

根据 Minijava.g4 的语法，我们根据 antlr4 的 rules 重写了 Minijava.g4 的内容。

2. 由 antlr4 生成的 lexer\parser\listener\visitor.java 文件，使我们可以在这个基础之上，查看生成的语法树。可以通过命令行的形式，也可以通过 java 重写（我们实现了两种方式）。

### 3. UI 部分

UI 调用命令行，做了 user-friendly 的处理。可以通过 Main.java 直接生成树，当然我们也可以仍然通过.bat 命令行的方式生成树（见 Read.me）。

4. 修改 g4 文件，实现错误检查。

G4 是 grammar 的定义部分。在 g4 中加入的扩充和检查都会被加入到之后的 lexer 和 parser 中。所以我们在 g4 中加入了：1. 更多的 tokens，比如之前没有定义的 import、comment 等；2. 错误修复：类名使用关键字等。

## 四、工作中遇到的问题

1. 我们在使用 antlr4 编译时，遇到了一些问题，我们发现根据 antlr4 安装指南，通过命令行无法输出我们想要的内容，并且会报错。

后来我们通过使用 .bat，以及我们发现需要在项目所在的文件夹下进行所有操作才可以。

2. 后来我们对 g4 的 rules 并没有那么了解，导致一开始我们并没有注意到 parser 和 lexer 的大小写问题，报出了语法左递归的错误，百思不得其解，还想去解决怎么左递归的问题。后来我们重新查看 antlr4 的规定，发现对 lexer 和 parser 有大小写的要求，于是我们重写了 g4 文件，跑通了。

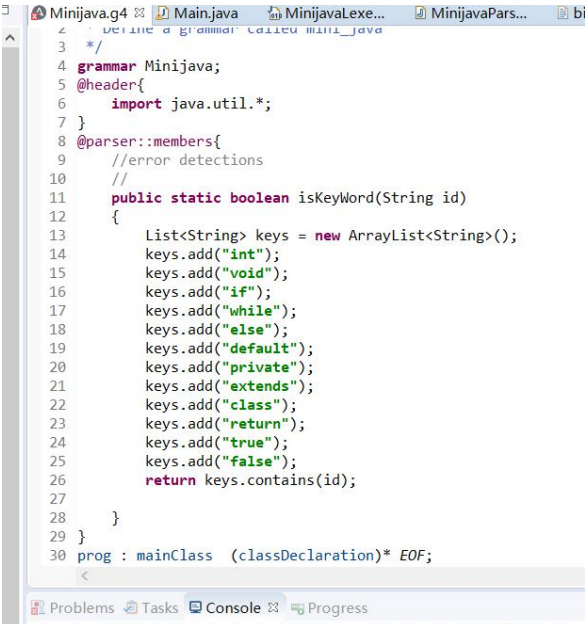
## 五、错误处理与修复

### 1. 注释行修复

我们在 g4 文件中加入了对于单行注释和多行注释的正则表达。因为这是在 MiniJava 的语法定义中没有出现，但是却是非常常见的一个语法现象。

### 2. 引用修复

同样地，我们在 g4 中加入了 import header，因为虽然是 MiniJava，我们还是需要有引用操作的。所以我们对 g4 再次进行了重写，加入了 @header 部分，允许代码中出现头部引用。



```
1  MiniJava.g4
2  grammar MiniJava;
3  @header{
4      import java.util.*;
5  }
6  @parser::members{
7      //error detections
8      //
9      public static boolean isKeyword(String id)
10     {
11         List<String> keys = new ArrayList<String>();
12         keys.add("int");
13         keys.add("void");
14         keys.add("if");
15         keys.add("while");
16         keys.add("else");
17         keys.add("default");
18         keys.add("private");
19         keys.add("extends");
20         keys.add("class");
21         keys.add("return");
22         keys.add("true");
23         keys.add("false");
24         return keys.contains(id);
25     }
26 }
27 prog : mainClass (classDeclaration)* EOF;
```

### 3. 类名使用关键字

当类名使用关键字的时候，系统需要报错。

我们在 g4 中增加函数如上页截图，函数中包括了我们可以想到的不可被重定义类名。

通过字符串比较的方式，在 g4 文件中增加函数，从而在 Parser 中可以生成检查。如果检查出错误信息，则会抛出系统报错。

错误：

```
class void {  
    }  
}
```

错误检测：

```
line 1:6 mismatched input 'void' expecting ID
```

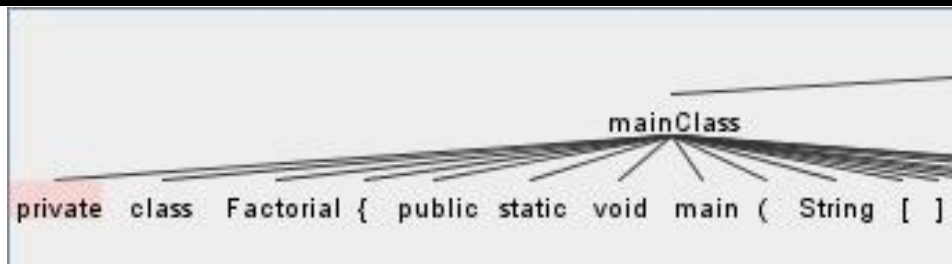
### 4. 错误声明处理

修改测试文件 factorial.java 的内容，比如声明 private class，通过 system.out.println 输出错误信息。

```
private class Factorial{  
    public static void main(String[] a){  
        System.out.println(new Fac().ComputeFac(10));  
    }  
}
```

命令行提示错误位置，树状图标红（如图所示）：

```
G:\Project\compile>java org.antlr.v4.gui.TestRig Minijava prog -gui factorial.tx  
t  
line 1:0 extraneous input 'private' expecting 'class'
```



这个错误信息同样可以在 UI 中显示。

## 5. 非法字符检测

当定义的代码内容中含有非法字符时，命令行会提示检测到错误字符。

错误：

```
class Fac {  
    ^  
    int x;  
}
```

错误检测：

```
line 2:4 token recognition error at: '^'
```

## 六、额外功能说明

### 1. user-friendly 的 UI 界面



我们的 Project 也可以使用代码而不是命令行来打印语法树。我们在代码中增加了 main.java，其中，我们对命令行做了调用。只要我们将要处理的文档输入，就可以输出语法树。示例如上图所示。

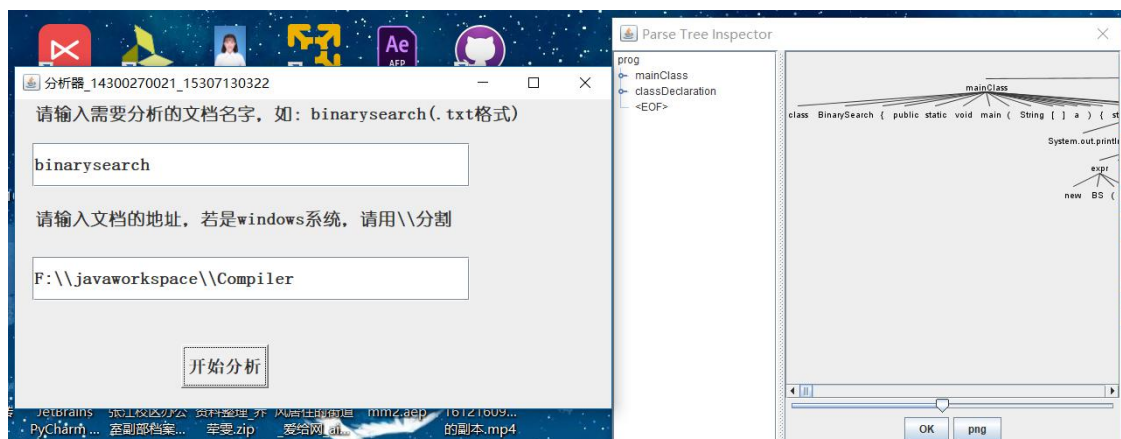
具体代码如下图：

```

4 import java.awt.*;
5 import java.awt.event.*;
6
7 import org.antlr.v4.runtime.*;
8 import org.antlr.v4.runtime.tree.*;
9
10 public class Main {
11     public static void main(String[] args) throws IOException{
12         EventQueue.invokeLater(() -> {
13             JFrame frame= new CreateFrame();
14             frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15         });
16     }
17 }
18
19
20 class CreateFrame extends JFrame implements ActionListener{
21
22     private Button ybt;
23     private JTextField tf1,tf2;
24     private JPanel ta1,ta2;
25
26     public CreateFrame(){
27         JFrame f = new JFrame();
28         f.setTitle("分析器_14300270021_15307130322");
29
30         ta1 = new JPanel();
31         ta1.setBounds(20, 0, 600, 30);
32         ta1.setBackground(new Color(0,0,0,0));
33         ta1.setFont(new Font("幼圆", Font.BOLD, 20));
34         ta1.setText("请输入需要分析的文档名字，如：binarysearch(.txt格式)");
35
36         tf1 = new JTextField();
37         tf1.setBounds(20, 50, 500, 50);
38         tf1.setFont(new Font("幼圆", Font.BOLD, 20));
39         tf1.addActionListener(this);
40
41         ta2 = new JPanel();

```

我们调用了 Java 的图形界面包，以及命令行操作，将原本需要在命令行中进行的编译操作转换到图形中。最后，我们可以获得的效果如下（同样的，如果有报错，在图形显示中也会显示出来）



## 七、项目感想

**乔萃雯：**

这个项目给我的感觉，又难又简单，每当我遇到 bug 的时候，我总是觉得很难解决，但我每次静下心来，好好查看文档或是查询网页，我最后总是可以把问题解决。所以我感觉这次的收获很大，我通过实际的操作，对编译的原理以及在实战中的应用有了深刻的了解~

**顾瑞珏：**

初看 Antlr 觉得是个功能很强大的编译器前端了，因为本身可以实现很多错误的位置标明和错误提示。但是当需要具体实现很多功能的时候文法文件就需要更加复杂的结构，在实现错误识别和修复过程中我也查询了很多资料，去看了作者提供的书籍，学习到了很多技巧和知识。这次的 project 帮助更好的理解了编译原理并进行了实践。