JML Report

余一夫(13307130355)

1. ANTLR 4

ANTLR 4 是一个基于 LL 算法的语法解析器及生成器，使用 Java 编写。考虑到本次项目的要求，及 TA 的推荐，便选择了 ANTLR 4。相比于其他的如 Yacc 和其他的工具，ANTLR 一大优势是使用 java 编写，构建的 jar 包能够在不同的系统上运行，减少了跨平台运行的难度，同时其也能生成 java 及其他对应语言的目标解析器，虽然在试用了 Go 的目标解释器后由于是 15 天前新出的语言，文档补全导致最终放弃了使用 Go 而选择使用 Java 进行编译器编写，但是支持语言的齐全本身也是选择 ANTLR 4 完成本次项目的原因之一。

2. 代码说明

本次所有代码均位于 moe.eve.jml 包下，最终生成的 jar 包的主类为 moe.eve.jml.MiniJava，使用的 ANTLR 版本为 4.6，代码主要分为 4 部分，其中包含入口类 MiniJava，类型实现类*Impl，包括 ClassImpl（类的实现），MethodImpl（类中方法的实现），TypeImpl（类型的实现）和 VarImpl（变量的实现），语法扫描类*Pass，包括第一次定义扫描 DefPass 和第二次检查扫描 CheckPass，剩余代码为 ANTLR 基于 minijava.g4 生成的解释器部分，以 miniJava 开头。

语法树生成主要分为两部分，第一部分为定义扫描，主要完成的工作是对所有的类，方法，及其其中的变量/类型进行逐一扫描登记。为了避免循环扩展，我对语法做了一定的限制，即使用 extend 扩展基类时，基类必须要在扩展类之前完成定义。由于不可避免的会出现 A 中出现 B 类的引用，同时 B 中出现 A 类的引用，所以无法一次扫描就完成判断是否所有变量都已经完成定义。故需要第二次检查扫描，他会对第一次扫描中定义的所有变量及方法中进行类型检查，以保证所有定义的变量或方法使用的类型都是已经定义的。

ANTLR 中其实是存在一些第一次使用不容易注意到的坑的，比如如果使用'+' exp | '-' exp 进行语法解析时，扫描到这一条语法 ANTLR 不能告诉我们匹配到的是可选部分的前半规则或者是后半规则，这也直接导致我最终没有办法完成语义分析。解决的办法是对所有的可选分支均进行命名，然后写不同的函数进行分析。但是改动语法文件不可避免的要导致整个项目的重写，鉴于时间因素无法重写，也算是本次项目的一大遗憾吧。

3. 错误处理

词法和语法错误的处理，ANTLR 已经做的相当完善了，能够在发生错误的地方提示检测到了错误的字符，并能给出对应的解决方案，如果是缺少特定的 token，还能提示需要补上哪些 token 才能使词法和语法变得正确。

而语义部分，我主要处理了两个方面:

第一方面为出现循环引用。循环引用出现分为两种，第一种为自己扩展自己，解决的方式为在记录父类时检查父类是否为自己；而第二种为多个类出现了一个扩展环，即 A extends B, B extends C, C extends A，对此我的解决方案是，当一个类作为基类被其他类扩展时，此时基类必须完成定义，这样就能避免扩展环的出现。

```
C:\Users\Eve\IdeaProjects\jml\out\artifacts\jml_jar>java -jar jml.jar -nogui
class Main { public static void main(String[] args) { System.out.println(1); } }
class A extends B { }
class B extends A { }
^Z
line 2:16 error: Class B not defined.
```

第二方面为出现重复定义。重复定义出现时，常常伴随出现的问题为不知道前一次定义的位置，故在 JML 中我记录了每一个定义的类、方法及变量的定义位置。当出现重复定义错误时，不仅显示发生错误的位置，同时提示最开始定义时的位置，能够帮助用户更好的定位错误。

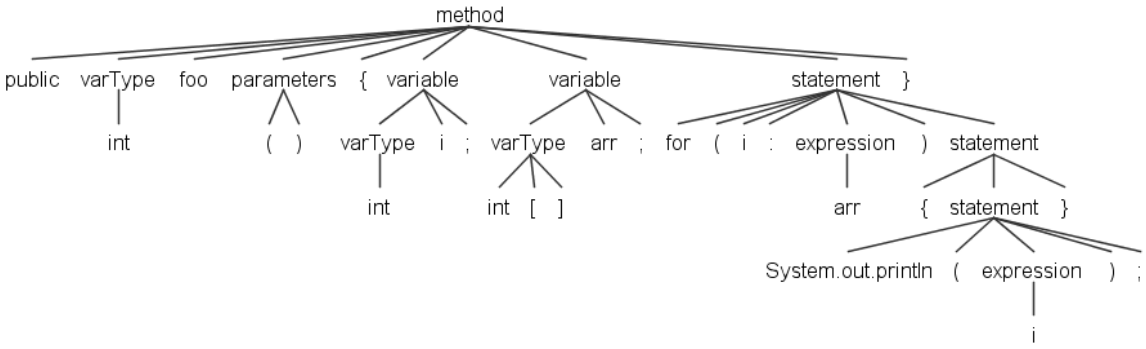
```
C:\Users\Eve\IdeaProjects\jml\out\artifacts\jml_jar>java -jar jml.jar -nogui
class Main { public static void main(String[] args) { System.out.println(1); } }
class A { }
class A {
}

^Z
line 3:0 error: Class A already defined
line 2:0 notice: Class A defined here.
```

```
C:\Users\Eve\IdeaProjects\jml\out\artifacts\jml_jar>java -jar jml.jar -nogui
class Main { public static void main(String[] args) { System.out.println(1); } }
class A {
 int foo;
 int bar;
 int baz;
 int foo;
}

^Z
line 6:1 error: Field foo already defined.
line 3:1 notice: Field foo defined here.
```

```
C:\Users\Eve\IdeaProjects\jml\out\artifacts\jml_jar>java -jar jml.jar -nogui
class Main { public static void main(String[] args) { System.out.println(1); } }
class A {
 public int foo() { return 1; }
 public int bar() { return 2; }
 public int foo() { return 3; }
}

^Z
line 5:1 error: Method foo already defined.
line 3:1 notice: Method foo defined here.
```

4. 项目感想

由于众多原因，我正式开始编写本项目的时间十分的晚，真正能够编程的时间不超过 48 小时，这也直接导致我本次项目完成度较低，许多希望实现的功能最后都因为时间原因没有能够实现，包括对语法的扩展，以支持 do-while 循环、for 循环和 foreach 循环，虽然生成的语法文件最终能够接受这样的输入，但是由于没有实现类型检查，最终的效果也十分的差。


method
public
varType
int
foo
parameters
( )
{
variable
varType
int
i
;
variable
varType
int [ ]
arr
;
statement
for
(i : expression)
arr
statement
{ statement }
System.out.println (expression) ;
i
}