MiniJava 编译器项目报告

罗齐尧 15307130105

吴灵杰 15307130091

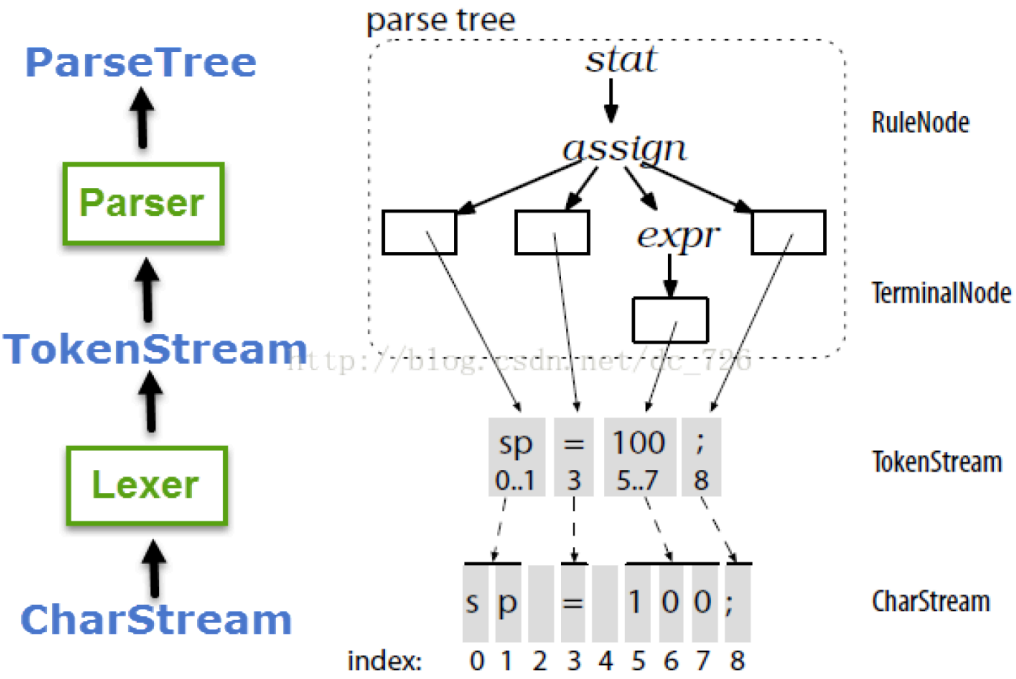
一、 工具选择 ANTLR

ANTLR v4 是一款可以用来阅读，处理执行或转化成结构的文本二进制件强大解析生成器。它在学术界和工业被广泛地用来建立各种语言，工具和框架。

如果给你的 ANTLR 生成的解析器一个有效输入，无论多么复杂语法都将正确识别生成的解析器一个有效输入，无论多么复杂语法都将正确识别 ，并能够自动地帮助你完成词法分析和语的工作。MiniJava 的 BNF 可以很轻易的改写成 ANTLR 支持 的.g4 语法文件。另外， ANTLR 拥有能够处理直接左递归文法、提供词处理直接左递归文法、提供词语法错误识别、树生 成、提供遍历语法树的类等优点。

综上，我们选择了 ANTLR 作为我们的工具。

二、 分析过程



ANTLR 会根据.g4 语法文件自动生成对应语言的解释器，词法分析和语法分析分别对应.g4 文件中的词法规则和语法规则。

简单起见，我们将解析分为两个阶段，对应我们大脑读取文字时的过程。当我们读到一个句子的时候，在第一阶段，大脑会下意识地将字符组成单词，然后像查字典一样识别出它们的意思。在第二阶段，大脑会根据已识别的单词去识别句子的结构。第一阶段的过程就是词法分析（lexical analysis），对应的分析程序叫做 lexer，负责将符号（token）分组成符号类（token class / token type）。第二阶段就是 parser，默认 ANTLR 会构建出一棵分析树（parse tree）或语法树（syntax tree）。

三、 代码框架

本次项目使用 ANTLR v4 和 Python 完成，另外需要的 Python 库有：antlr4-python3-runtime 和 graphviz。

```
---- src                                  // 代码文件夹
-------- MiniJava.g4                      // 语法文件
-------- MiniJava.py                      // 词法、语法检查，输出抽象语法树
-------- MiniJavaLexer.py                 // 文法检查
-------- MiniJavaParser.py                // 词法分析
-------- MiniJavaListener.py              // 监听所有标识符
-------- MiniJavaVisitor.py               // 遍历整棵语法树
-------- MiniJavaErrorHandler.py          //  user-friendly 错误提示
-------- MiniJavaSemanticHandler.py       // 语义检查
```

核心代码包括：

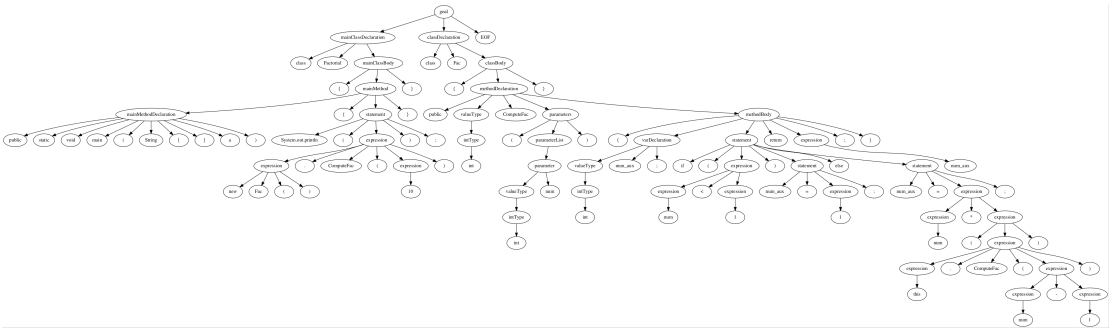
1. 语法、词法检查，这部分利用 ANTLR 命令完成
2. 语义检查，利用 listener 遍历整个树，后文会具体说明
3. 输出语法树，利用 Python Graphviz 实现可视化

样例 factorial.java：

factorial.java

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

class Fac {

    public int ComputeFac(int num){
    int num_aux ;
    if (num < 1)
        num_aux = 1 ;
    else
        num_aux = num * (this.ComputeFac(num-1)) ;
    return num_aux ;
    }

}

```

```
luoqiyao@SixSiebenUno-MBP ~/Documents/GitHub/MiniJava/src master python MiniJava.py ../sample/factorial.java -AST -treegraph
----------------------
Lexical and Syntax Check...
Lexical and Syntax Check Done.
----------------------
Semantic Check...
Semantic Check Done.
----------------------
AST string:
(goal (mainClassDeclaration class Factorial (mainClassBody { (mainMethod (mainMethodDeclaration public static void main ( String [ ] a )) { (statement System.out.println ( (expression (expression new Fac
( )) . ComputeFac ( (expression 10) )) ) ;) }) (classDeclaration class Fac (classBody { (methodDeclaration public (valueType (intType int)) ComputeFac (parameters ( (parameterList (parameter (valueTyp
e (intType int)) num)) )) (methodBody { (varDeclaration (valueType (intType int)) num_aux ;) (statement if ( (expression (expression num) < (expression 1)) ) (statement num_aux = (expression 1) ;) else (s
tatement num_aux = (expression (expression num) * (expression ( (expression (expression this) . ComputeFac ( (expression (expression num) - (expression 1)) )) ))) ;)) return (expression num_aux) ; })) }))
 <EOF>)
----------------------
Generate Parse Tree...
Generate Parse Tree Finished.
```



四、 错误处理和修复

1. 词法错误

factorial.java

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

class Fac {

    public int ComputeFac(int num){
    int num_aux ;
    if (num < 1)
        num_aux = 1 ;
    elsse // should be else
        num_aux = num * (this.ComputeFac(num-1)) ;
    return num_aux ;
    }

}
```

```
luoqiyao@SixSiebenUno-MBP ~/Documents/GitHub/MiniJava/src master • python MiniJava.py ../sample/factorial.java
------------------------
Lexical and Syntax Check...
line 13:1 missing 'else' at 'elsse'
Lexical and Syntax Check Done.
------------------------
Semantic Check...
line 13:1       SemanticFault Unknown Identifer: elsse
        elsse // should be else
        ^
Semantic Check Done.
------------------------
```

2. 语法错误

factorial.java

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

class Fac {

    public int ComputeFac(int num){
    int num_aux ;
    if (num < 1)
        num_aux = 1 ;
    // missing else
        num_aux = num * (this.ComputeFac(num-1)) ;
    return num_aux ;
    }

}

```

```
-----------------------
luoqiyao@SixSiebenUno-MBP ~/Documents/GitHub/MiniJava/src master • python MiniJava.py ../sample/factorial.java
-----------------------
Lexical and Syntax Check...
line 14:5 missing 'else' at 'num_aux'
Lexical and Syntax Check Done.
-----------------------
Semantic Check...
Semantic Check Done.
-----------------------
```

3. 语义错误

这里的语义错误检查包括了：变量未定义、变量重定义、类重定义。

factorial.java

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

class Fac {

    public int ComputeFac(int num){
    int num_aux ;
    int num_aux ;
    a = 1;
    if (num < 1)
        num_aux = 1 ;
    else
        num_aux = num * (this.ComputeFac(num-1)) ;
    return num_aux ;
    }

}

class Fac {

    public int ComputeFac(int num){
    return 0
    }

}
```

```
luoqiyao@SixSiebenUno-MBP ~/Documents/GitHub/MiniJava/src master • python MiniJava.py ../sample/factorial.java
-----------------------
Lexical and Syntax Check...
line 26:4 missing ';' at '}'
Lexical and Syntax Check Done.
-----------------------
Semantic Check...
line 11:5       SemanticFault Multiple Varations Dec: num_aux
        int num_aux ;
            ^
line 12:1       SemanticFault Unknown Identifer: a
        a = 1;
        ^
line 22:6       SemanticFault Multiple Class Dec: Fac
class Fac {
      ^
Semantic Check Done.
-----------------------
```

具体的实现方法是：遍历一遍语法树，遍历同时维护一个 hash table，记录每个 ID 以及他们属于类或者变量。当我们遇到一个 expression 就检查这个 ID 是否在当前域内有定义，如果没有定义就输出未定义的 ID 和类型。在赋值阶段检查是否有重定义的现象。这样就完成了语义检查。

4. 错误修复

这里利用了 ANTLR 自带的错误修复功能，当遇到相应的错误的时候，对正确结果进行猜测，通过这种方式，不会影响到下面语法分析步骤的进行，实现了相应的错误修复功能。

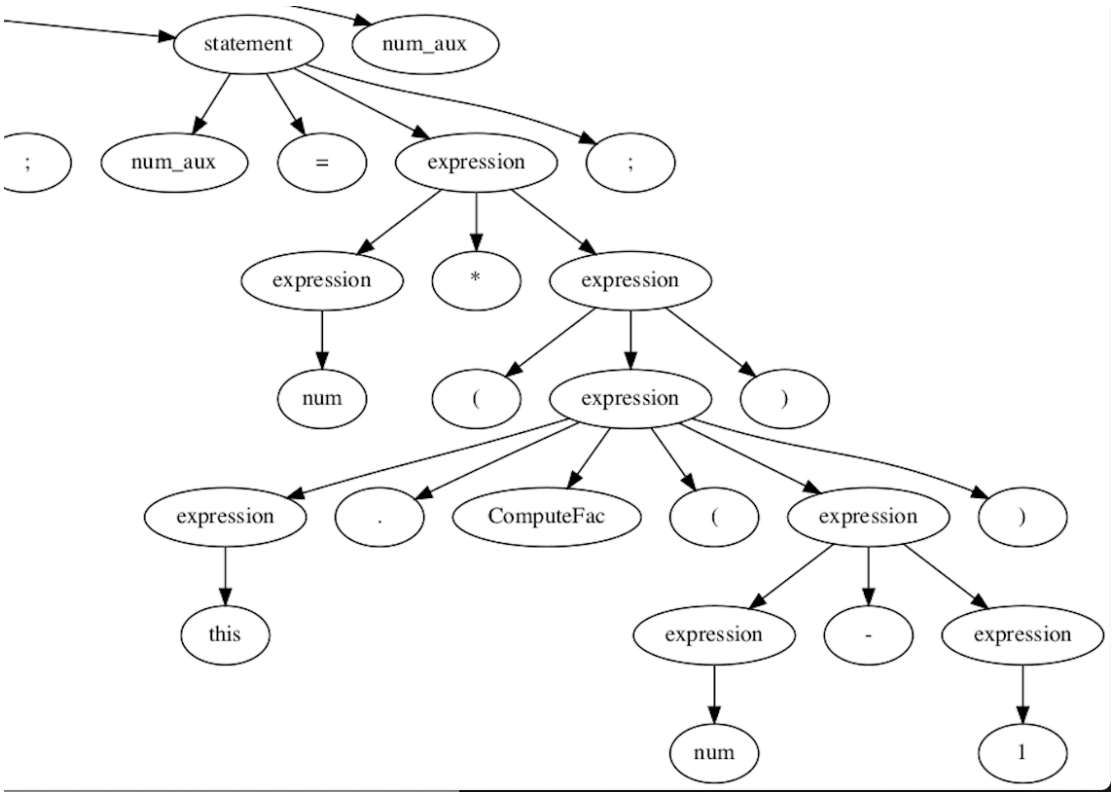
五、可视化抽象语法树

文字版本的 AST，利用了 ANTLR 自带的 toStringTree()函数完成。

由于 ANTLR 的 python 运行库不支持 GUI，所以自行完成了树图的绘制，通过遍历整棵语法树，建立所有的节点和边，通过生成 dot 文本，使用 Graphviz 的库函数，展示可视化图像。

之前样例中的图片细节展示如下，对应语句：

num_aux = num * (this.ComputeFac(num-1)) ;



六、 总结

罗齐尧： 本次 PJ 依赖 ANTLR 完善的机制进行，使用 Python 作为 ANTLR 开发的难点在于参考资料过少，主要还是 Java 语言的版本，不过由于接口的共同性还是可以直接通过阅读源代码找到相应的方法。其中最难的就是在语义错误检查阶段，需要重写一个访问函数，但是由于 ctx 等变量的意义不明，导致多次尝试才完成了代码相应的检查。

吴灵杰： 编译 pj 让我了解了编译器对于高级语言的具体处理流程，接触到了更深层次的语言技术，让我获益良多。