Antlr 学习笔记

之 Antlr 工具学习篇

龚海兵

200732580046

国际软件学院 软工 2 班

2009-9-26

dylanninin@gmail.com

目录：

参考书目：antlr.pdf

Antlr 入门教材

网络资源

1.Antlr 安装(这里以 win32 为例)

1) antlrworks 下载

由于 antlr 是用 java 编写，安装 antlr 时，电脑上需要预先安装 jre1.4 或更高版本.

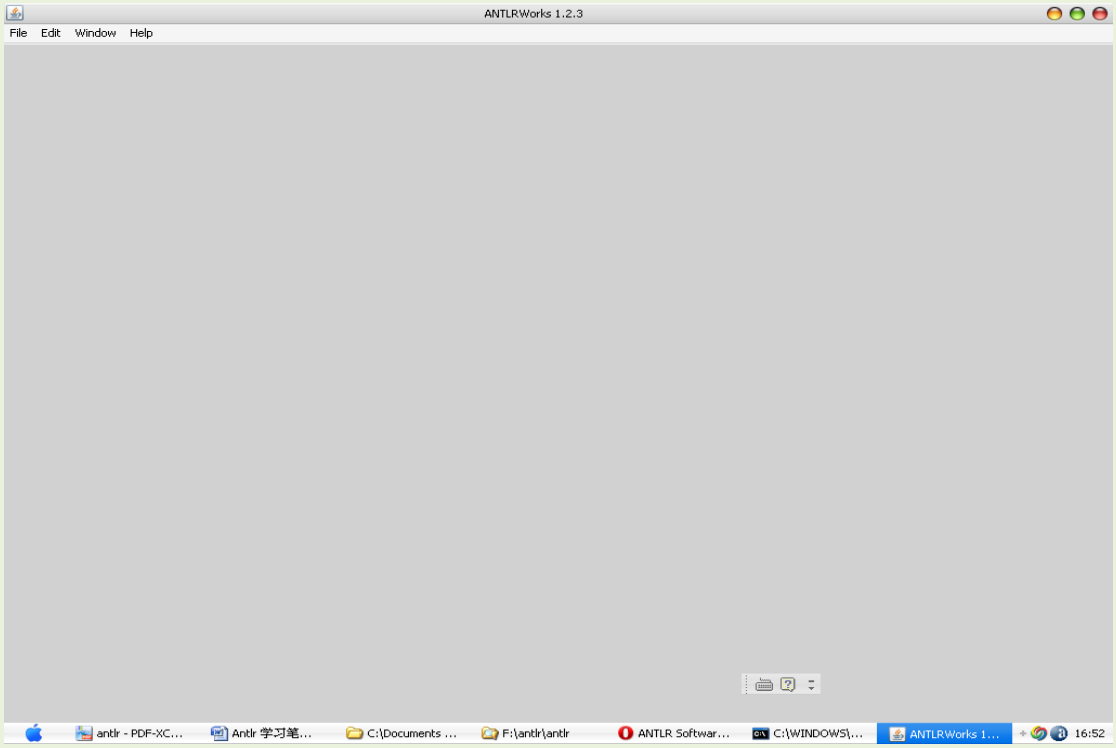
到 antlr 官网 http://www.antlr.org/download.html 下载 antlrworks-1.2.3.jar(注意 antlrworks 需要 jre1.5 以上)。

图 1-1【antlr 下载页面】

2)antlrworks 配置

在 antlr 安装目录下，双击即可以启动 antlrworks。

图 1-2【antlr 启动界面】



若想在命令行下启动 antlrworks，则需配置 **path: $antlr 安装目录$\;** 这样进入命令行后，在任何目录下，都可以输入 **antlrworks-1.2.3.jar** 来启动 antlrworks。如果嫌输入 antlrworks-1.2.3.jar 文件名太长，可以
该为相对短的名字，如 antlr.jar 以后输入相应名字即可.

编译 java 文件时，需要用到 antlr 运行时一些 jar 包，可以到官网下载 antlr-3.1.3.jar, 编译 antlr 生成的 java 代码时，可以用 **-classpath antlr-3.1.3.jar** 找到需要的类库；当然也可以在系统环境变量中配置 **classpath**，就免去了每次编译时添加 classpath 选项的麻烦。

2.Antlr 文法形式(以 E.g 为例)

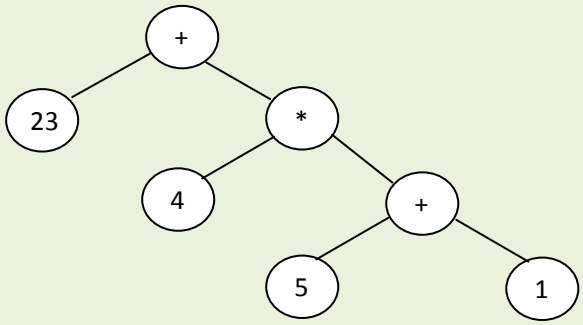
两个简单的表达式语句：

23+4*(5+1);　　str="Hello World";

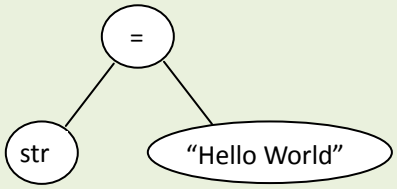
第一条语句是一个算术表达式，括号改变了运算顺序，计算结果不赋给任何变量。第二条是一个赋值表达式，将字符串赋给一个变量。

后面将要开发一个语法分析器来分析这两条语句。

23+4*(5+1)的语法树根据操作符的优先级如下:



赋值表达式的语法树结构：



赋值操作符“=”做为根节点变量 str 作为左子树，而字符串表达式“Hello World”作为右子树。

antlr 文法后缀名为.g，由于采用 ASCII 编码，也可以用记事本等工具打开。文法 E.g 如下

```
grammar E;

options{
   output=AST;
   }

program  :
   statement+;

statement:
   (expression | VAR '=' expression)';';


expression:
   (multExpr(('+'|'-')multExpr)*)|STRING;

multExpr:
   atom('*'atom)*;

atom  :
```

```
    INT | '('expression')';

INT   :
   '0'..'9'+;

VAR   :
   ('a'..'z' | 'A'..'Z')+;

STRING   :
   '"'( ('A'..'Z' | 'a'..'z' | ' ' | '0'..'9' )+)'"';

WS :
    (' ' | '\t' | '\n' | '\r'  ) + {skip();};
```

说明:

1)grammar:

grammar E 的 E 为文法的名称它与文件名一致.

2) options:

文法的设置部分

output=AST(Abstract Syntax Tree)表示让语法分析器返回包含语法树的信息。这里也可以进行目标语言设置.(如 language=CSharp 等)

ANTLR 在不指定目标语言的情况下默认是 java 语言。

其他还有 filter、greedy 等相关选项设置。

3)program:

文法是用 EBNF1 推导式来描述的（有关 EBNF 会在后面章节中讲解），文法定义中分两大部分以小写字母开头的语法描述和全大写的词法描述。其中每一行都是一个规则（rule）或叫做推导式、产生式，每个规则的左边是文法中的一个名字，代表文法中的一个抽象概念。中间用一个“:”表示推导关系，右边是该名字推导出的文法形式

4）statement

代表表达式语句

推导式中以“|”分隔代表“或”关系。表达式本身是合法的语句，表达式也可以出现在赋值表达式中组成赋值语句，两种语句都以“;”字符结束。

5)其他定义基础

连接“ ”：规则 A : a b c; a、b、c 之间用空格分隔。此规则接收句型 abc，符号 a、b、c 是按顺序连接起来的关系。

选择“| ”：规则 A : a | b | c; “|”表示“或”的关系，符号 A 可以推导出 a 或 b 或 c，也就是在 a、b、c 中选择。这要比写成 A : a; A : b; A : c; 方便得多。连接和选择可以联合起来使用，如 A : a b c | c d e;。有进也会使句型的数量增多如：A : B D; B : a | b; D : c | d;这时符号 A 推导出的句型有 ac、ad、bc、bd 四种。

重复“*，+”：规则 A : a*; “*”表示 a 可以出现 0 次或多次。A : a*; 相当于 A : A a | ;。这样可以避免递归的定义，可文法定义中递归往往引起文法的二义性。如果 a 至少要出现一次可以表示为 A : a+; “+”表示 a 可以出现 1 次或多次。相当于 A : A a | a;。重复可以和连接、选择一起使用如：A : a* b | c+ d;。

可选“？”：规则 A : a?; “?”表示 a 可以出现 0 次或 1 次，即 a 可有可无。相当于 A : a | ;。可选可以和连接、选择、重复一起使用如：A : a* b? | c+ d?;。

子规则“()”：规则 A : (a b) | b; a 与 b 在括号中，这样“(a b)”形成了一个子规则，也就是说可以把规则写成 A : B | b; B : a b;两个规则表示，我们把 B 规则用括号括起来放到 A 规则中这样就是 A 规则的子规则了。利用子规则也可以把多个符一起进行描述，A : (a b c)* 规则中 a、b、c 三个符号可以一起重复 0 次或多次。子规则有利于我们把很复杂的多个规则写到一起，有时这样写会使文法既简练又直观。子规则和前面的各种特性用到一起可以把复杂的文法写的很浓缩。如：A : (a b c)* | (c d)+ e?;.

值得注意的是如果我们的规则中有“()”的字符该如何表示？因为子规则也是用“()”表示的。在 ANTLR 中表示字符要用“’”单引号括起来，用‘(’ ‘)’来表示括号字符。前面讲到的表示文法规则的符号“| * + () ?”叫做文法的元符号。

skip()方法:有些字符是不属于源程序范畴内的，这些字符在分析过程中应该忽略掉。在 ANTLR 中可以在词法定义中加入 skip();，(如果是 C#为目标语言为 Skip();)。在规则的定义的之后与表示定义结束的分号之前加入“{skip();}”。例如下面定义了一个跳过空白的词法定义。

```
WS : ( ' ' | '\t' | '\n' | '\r' ) + {skip();}
```

3.Antlr 结构

1)antlr 主要类

Lexer： 文法分析器类。主要用于把读入的字节流根据规则分段。既把长面条根据你要的尺寸切成一段一段:）并不对其作任何修改。

Parser： 解析器类。主要用于处理经过 Lexer 处理后的各段。一些具体的操作都在这里。

2) antlr 主要工作流程

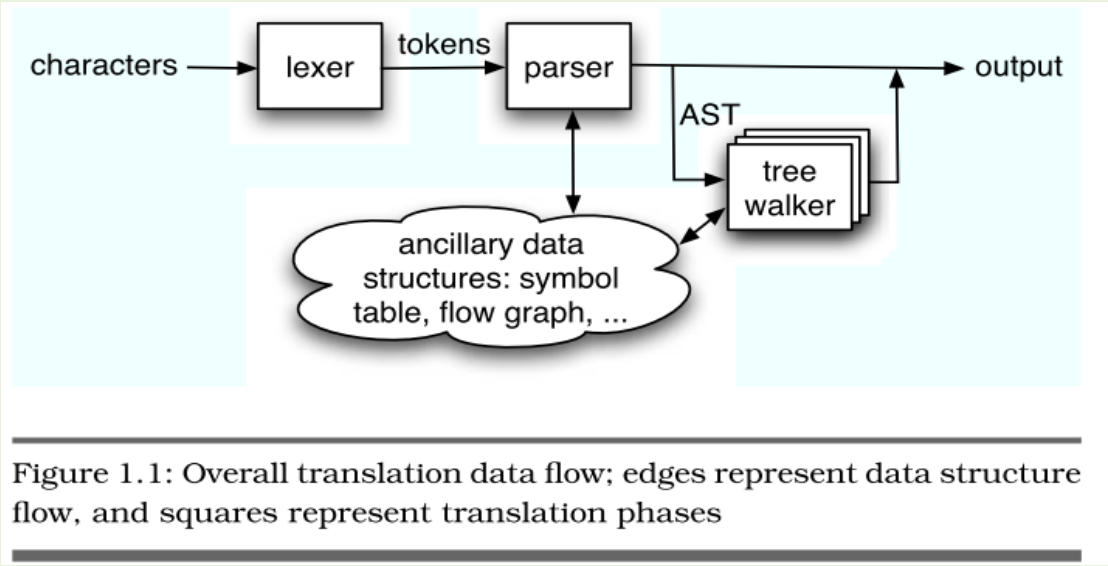


Figure 1.1: Overall translation data flow; edges represent data structure flow, and squares represent translation phases

antlr 运行时主要的数据类型

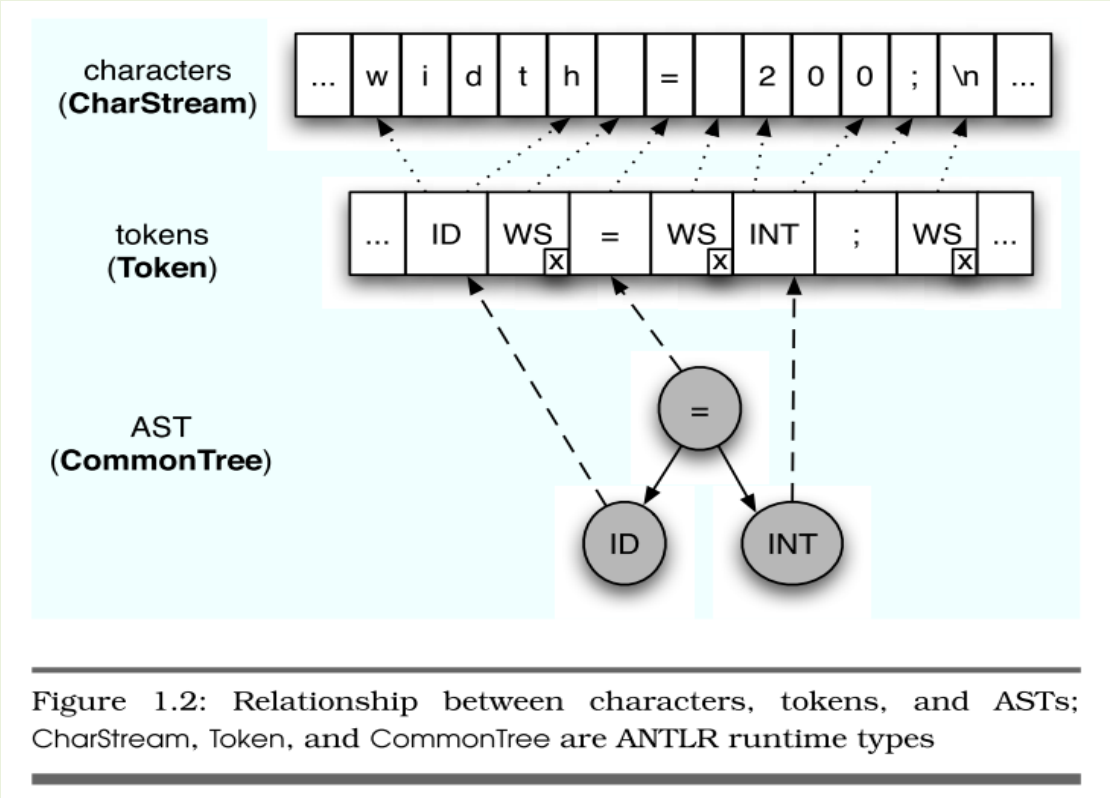


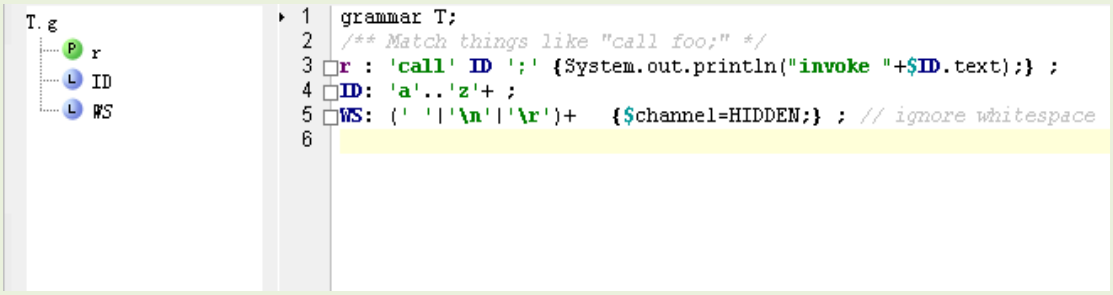
Figure 1.2: Relationship between characters, tokens, and ASTs; CharStream, Token, and CommonTree are ANTLR runtime types

4.Antlr 生成 java 练习

1）Chapter1_T (antlr.pdf chapter1 中示例)

语法 T.g 如下。

图 1-3【T.g 文法】



在 T.g 目录下，用 **java org.antlr.Tools T.g** 命令生成 **TLexer.java** 和 **TParser.java** 文件

在当前目录下，再编写一个测试 java 类 **Test.java**，即可以对简单的输入进行词法分析。

图 1-4【Test.java】

```
/***
 * Excerpted from "The Definitive ANTLR Reference",
 * published by The Pragmatic Bookshelf.
 * Copyrights apply to this code. It may not be used to create training material,
 * courses, books, articles, and the like. Contact us if you are in doubt.
 * We make no guarantees that this code is fit for any purpose.
 * Visit http://www.pragmaticprogrammer.com/titles/tpantlr for more book informatic
***/
import org.antlr.runtime.*;

public class Test {
    public static void main(String[] args) throws Exception {
        // create a CharStream that reads from standard input
        ANTLRInputStream input = new ANTLRInputStream(System.in);

        // create a lexer that feeds off of input CharStream
        TLexer lexer = new TLexer(input);

        // create a buffer of tokens pulled from the lexer
        CommonTokenStream tokens = new CommonTokenStream(lexer);

        // create a parser that feeds off the tokens buffer
        TParser parser = new TParser(tokens);
        // begin parsing at rule r
        parser.r();
    }
}
```

用 javac 命令编译 **java** 类

javac Test.java

再用 java 执行 Test

java Test

```
F:\antlr\works\java\Chapter1_T\output>java Test
call foo;
^Z
invoke foo

F:\antlr\works\java\Chapter1_T\output>_
```

```
F:\antlr\works\java\Chapter1_T\output>java Test
call foo
^Z
line 0:-1 missing ';' at '<EOF>'
invoke foo

F:\antlr\works\java\Chapter1_T\output>_
```

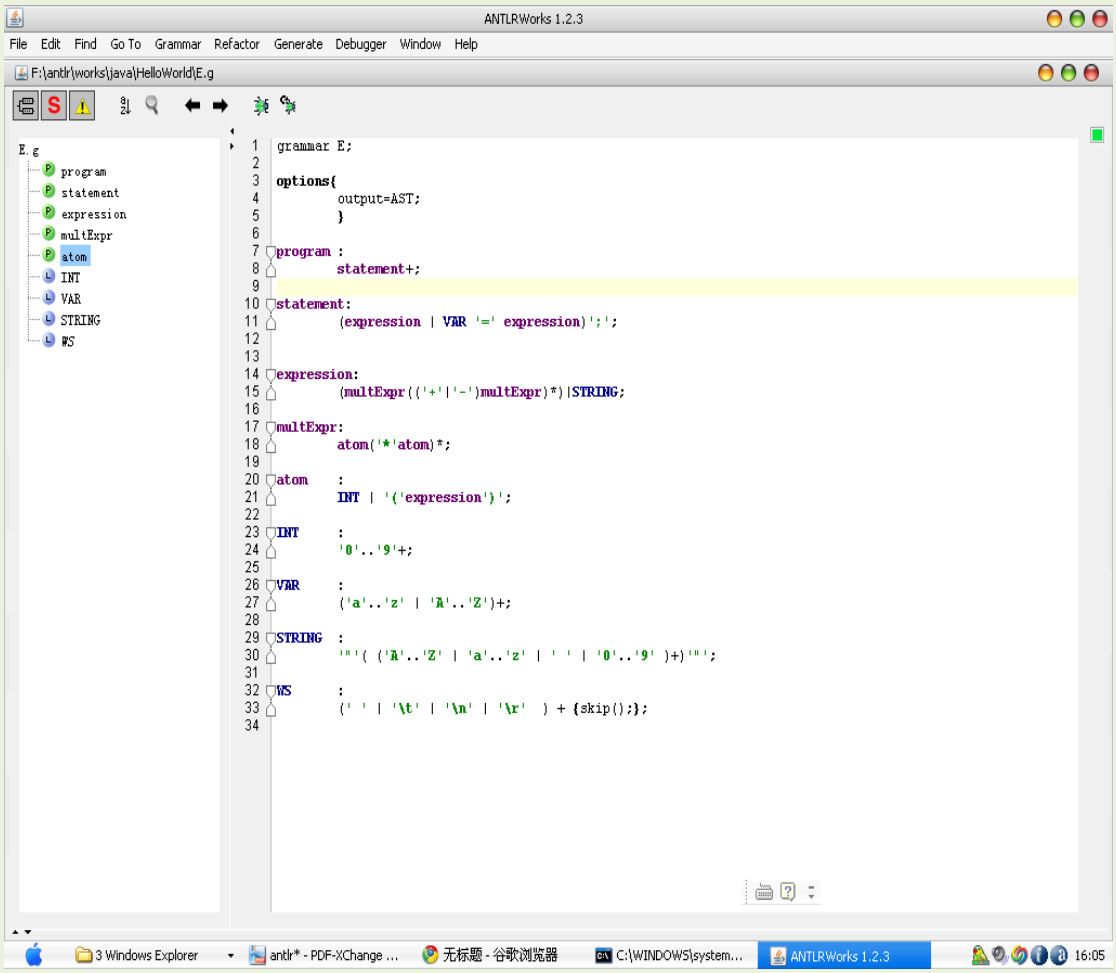
注意输入完待分析的字符串后，要 **enter** 换行，以作为输入结束的标志，再按 **ctrl + Z** 开始对输入进行分析。若无结束标志，则程序会永远处于待输入状态。

```
F:\antlr\works\java\Chapter1_T\output\bin>java Test
call string;^Z
cal;^Z
dsd:
^Z
line 1:12 no viable alternative at character '→'
line 2:4 no viable alternative at character '→'
line 3:3 no viable alternative at character ':'
invoke string

F:\antlr\works\java\Chapter1_T\output\bin>_
```

2）HelloWorld （简单算法和字符串表达 示例）

语法 **E.g** 如下

图 1-5【E.g 文法】



在生成 **java** 代码后，编写 **Run.java** 作为测试类。

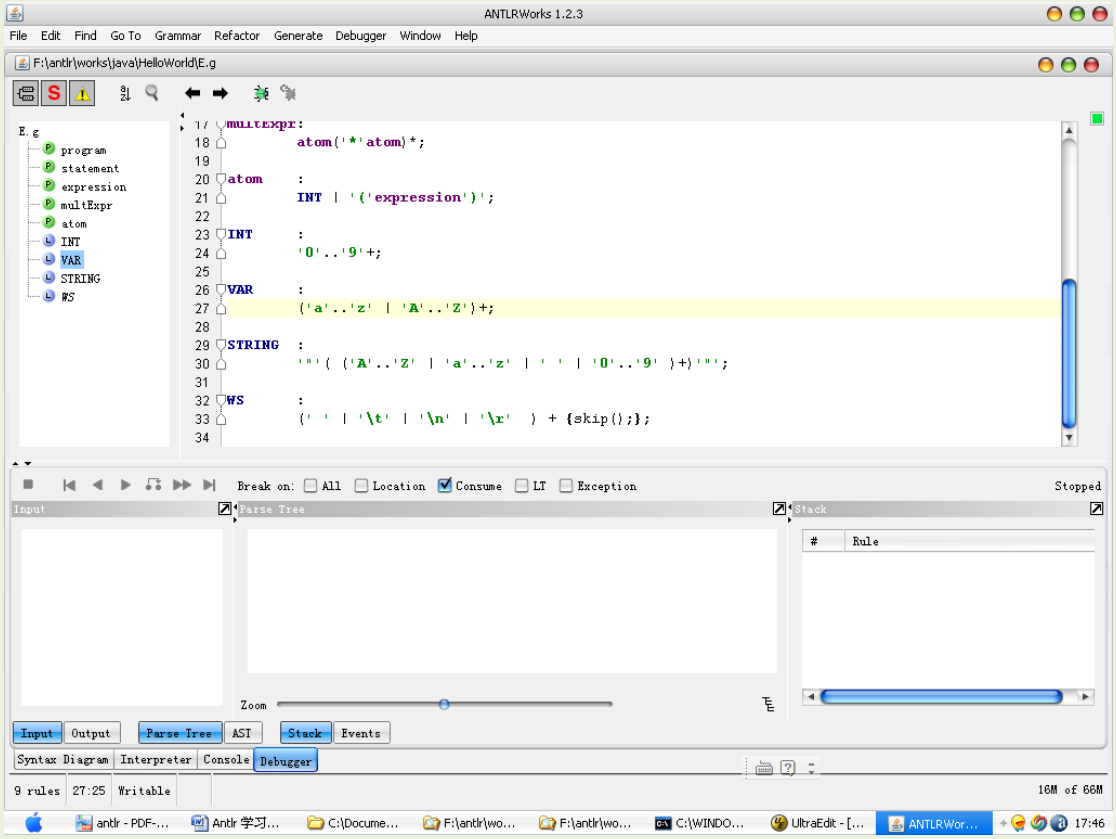
图 1-6【Run.java】

```
import org.antlr.runtime.*;
import org.antlr.runtime.tree.*;

public class Run{

   public static void main(String[] args)throws Exception {

      /**
        对输入输出流进行包装
      */
      ANTLRInputStream input = new ANTLRInputStream(System.in);

      ELexer lexer = new ELexer(input);
      CommonTokenStream tokens = new CommonTokenStream(lexer);
      EParser parser = new EParser(tokens);
      EParser.program_return r = parser.program();//program 为文法的启点

      System.out.println( ((BaseTree)r.getTree()).toStringTree());

   }
}
```

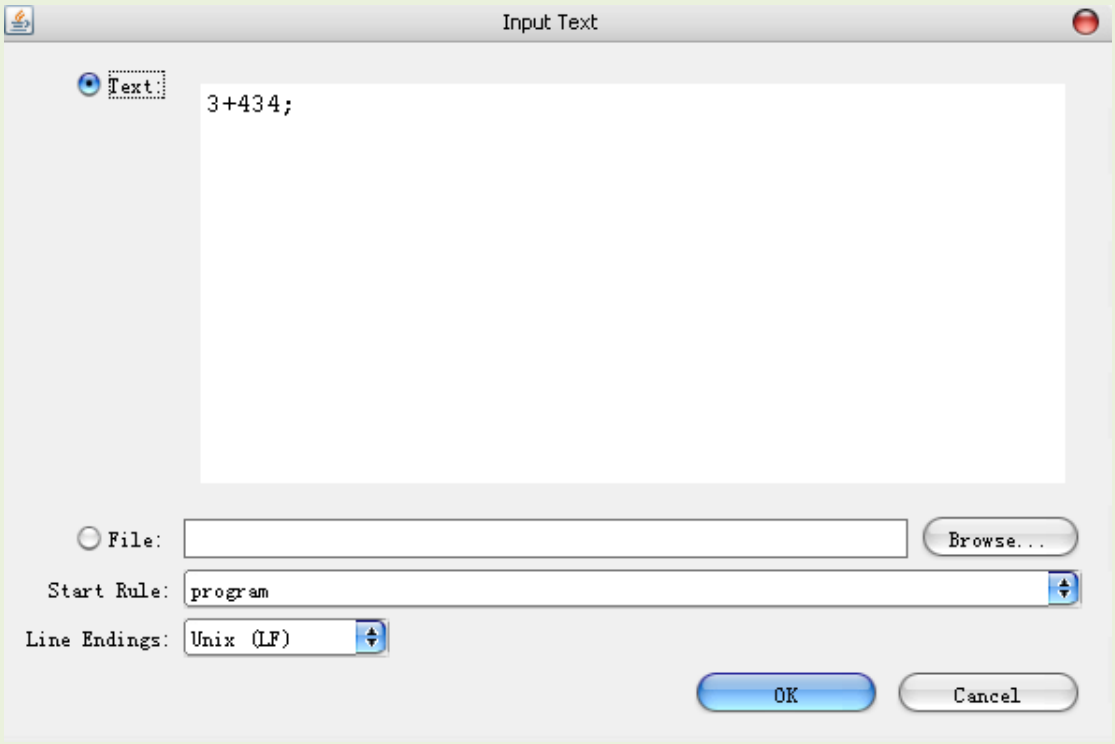
编译和解释执行如上，输入正确则会得到相应的树，否则提示错误信息，这里不再赘述。

5.Antlr Debug （以 HelloWorld 为例）

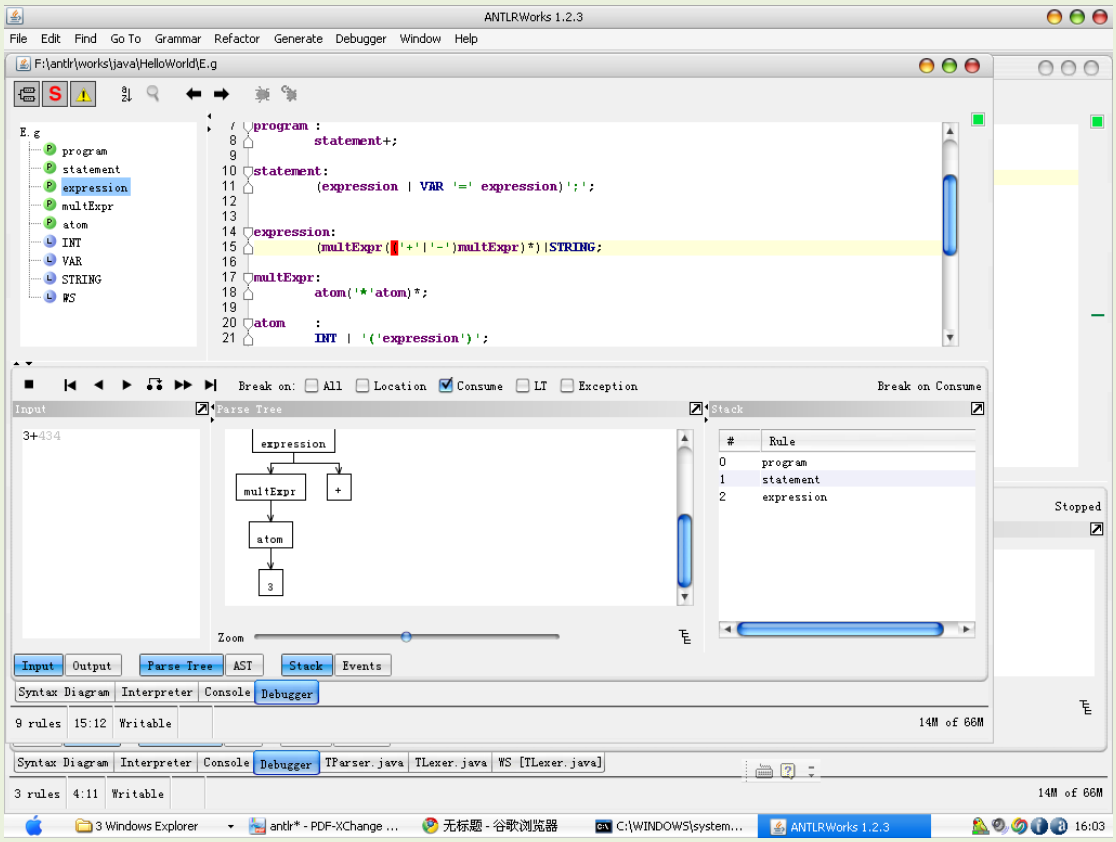
1)antlrworks debug 界面



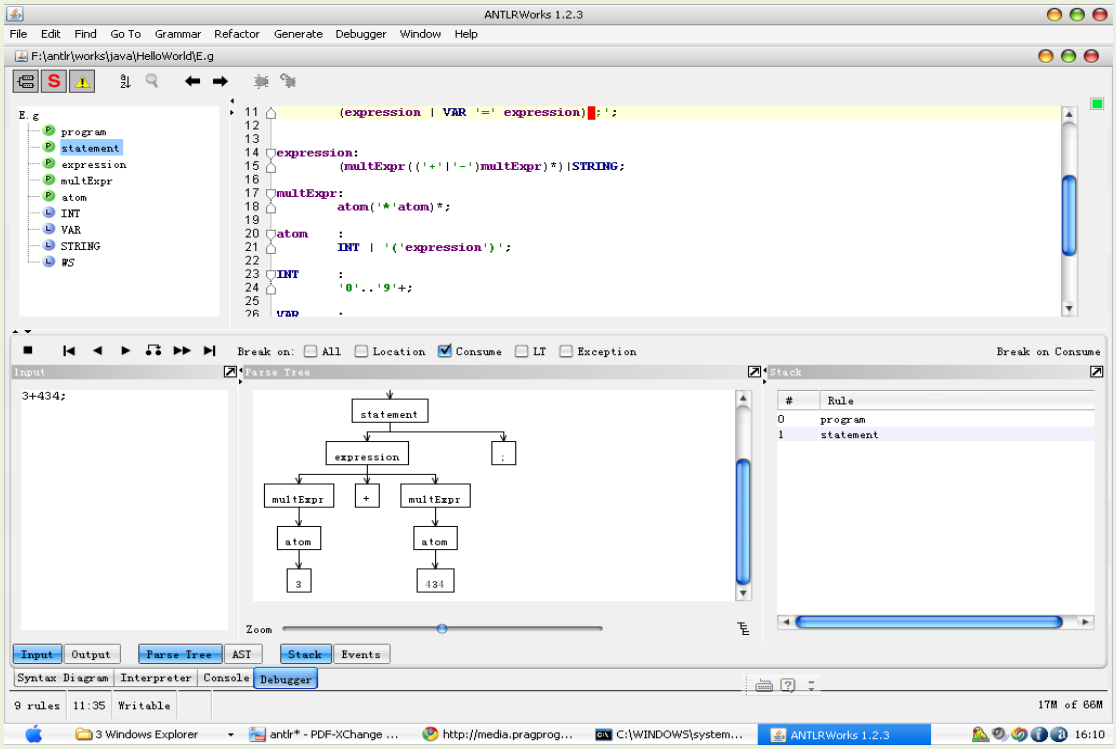
在选择 debug 菜单时，会有 input 对话框，提示输入测试数据，这里以输入"3+434;"为例。



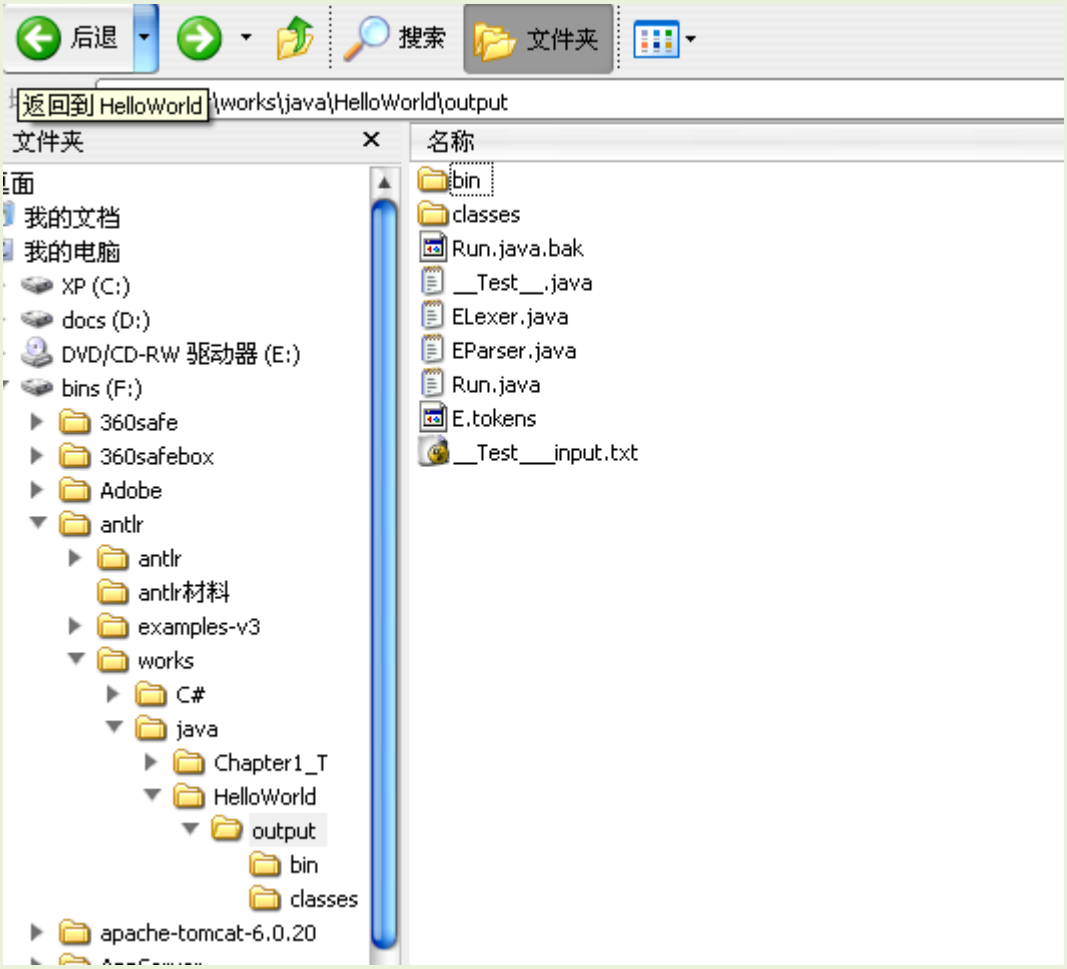
点击 ok 按钮，即可以进行 debug。



在 debug 过程中，请注意 input ,parser tree 和 stack 窗口的变化。它们之间是同步的。



在 debug 之后，在当前目录下，会产生 debug 相关文件：



其中以"_"开始和结尾时产生的测试类(_Test_.java)和测试数据(_Test_input.txt)；这些文件起到了测试日志的作用，很大的方便了程序测试的记录工作。

classes 是 debug 生成的 class。

6.Antlr Calc 简单实现

到这里，对 antlr 的基本使用已经有大概的了解和掌握。下面进行 Calc 简单计算器词法语法分析的自动实现做实验。

1)实验简介：

实验目的：进一步熟悉 antlr IDE 的使用，掌握 antlr 基本文法规范。

实验用例：正确解析带有括号的数学四则混合运算表达式,如 4+34;(4-5*(94.88-4)/5.5);等等

2)文法设计

```
grammar Calc;

options{
    output=AST;
```

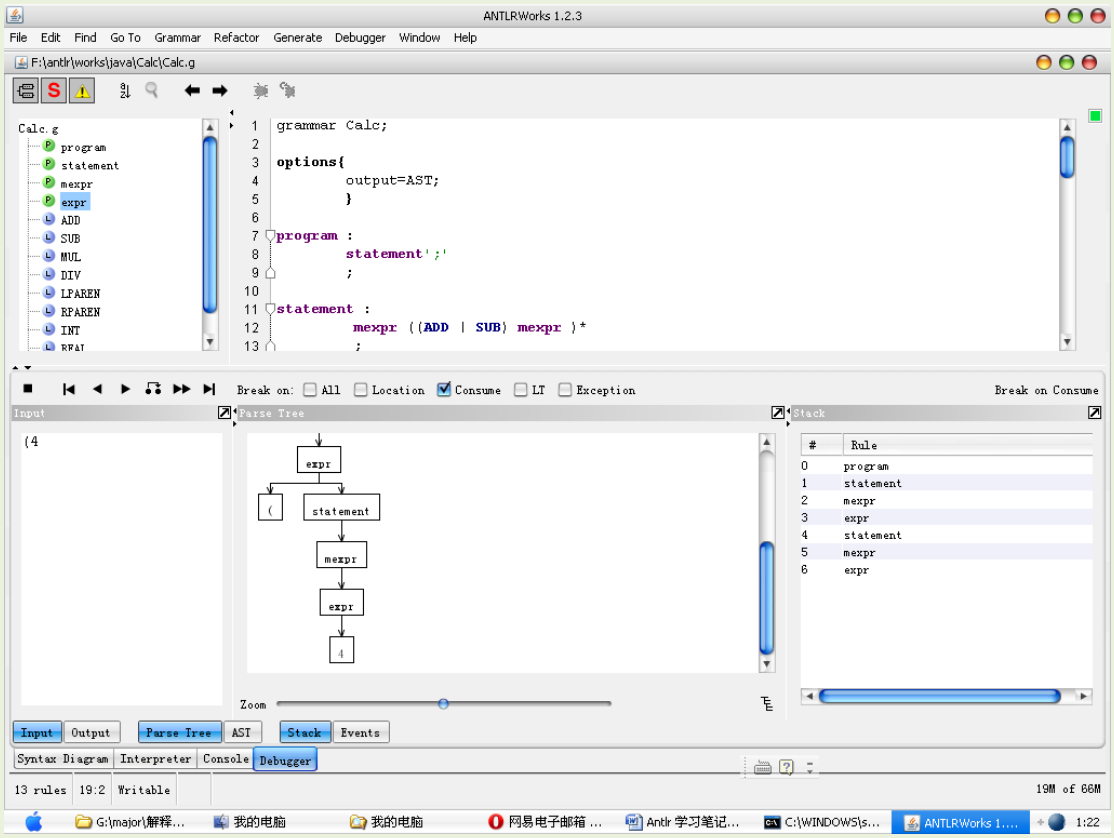
```
    }
program :
    statement';'
    ;

statement :
     mexpr ((ADD | SUB) mexpr )*
     ;
mexpr : expr ( (MUL | DIV) expr )*
    ;
expr :
    INT | REAL | LPAREN statement RPAREN
    ;

ADD :
    '+'
    ;
SUB :
    '-'
    ;
MUL :
     '*'
    ;
DIV :
     '/'
    ;
LPAREN  :
    '('
    ;
RPAREN  :
    ')'
    ;
INT :
    '0'..'9'+
    ;
REAL    :
    '0'..'9'+('.''0'..'9'+)?
    ;

WS  :
    (' ' | '\t' | '\n' | '\r'  ) + {skip();}
    ;
```

3)代码生成和测试

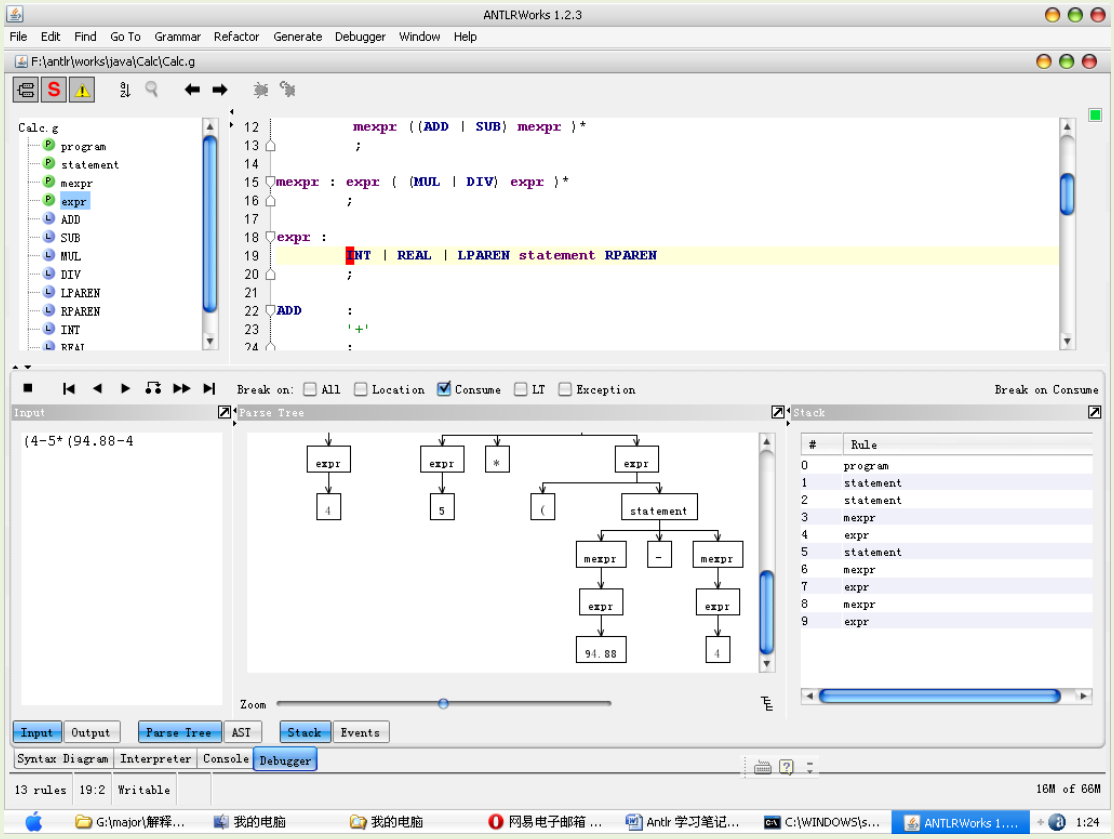
生成 java 目标代码后，debug，在 input 对话框中输入”（4-5*(94.88-4)/5.5);”进行调试

调试过程中几张截图如下：

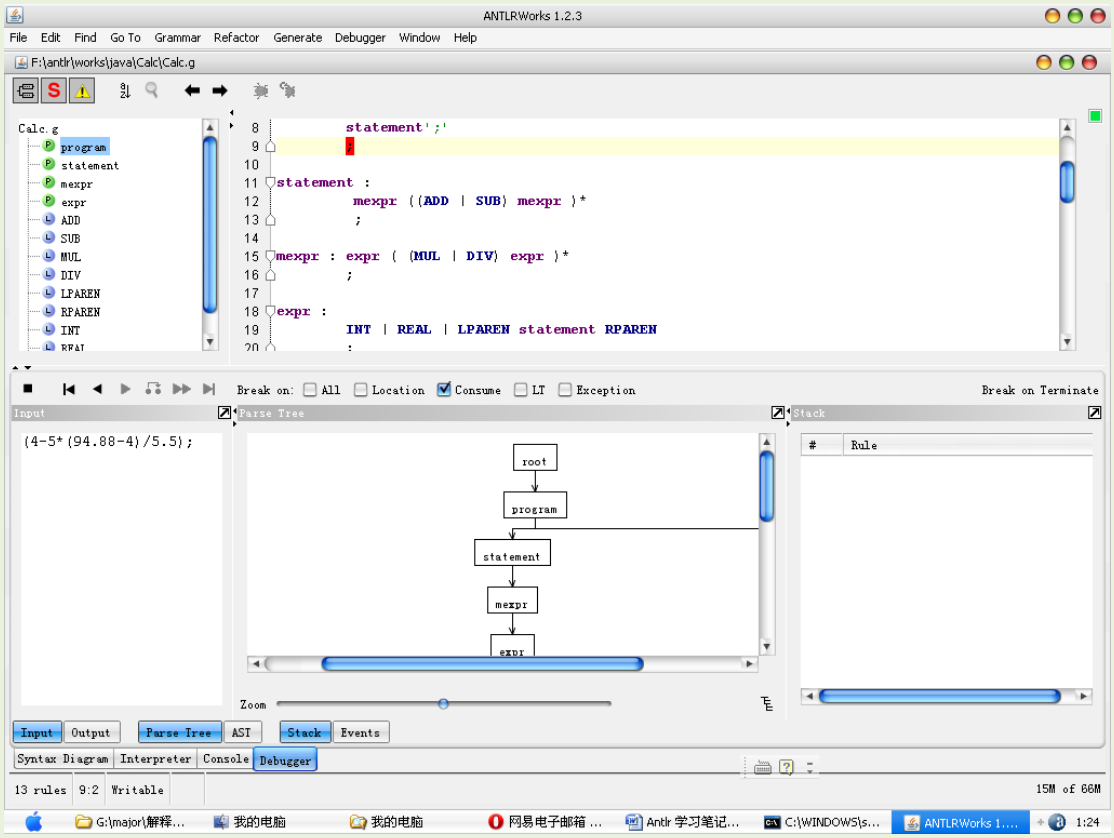
a.执行到“(4”时：



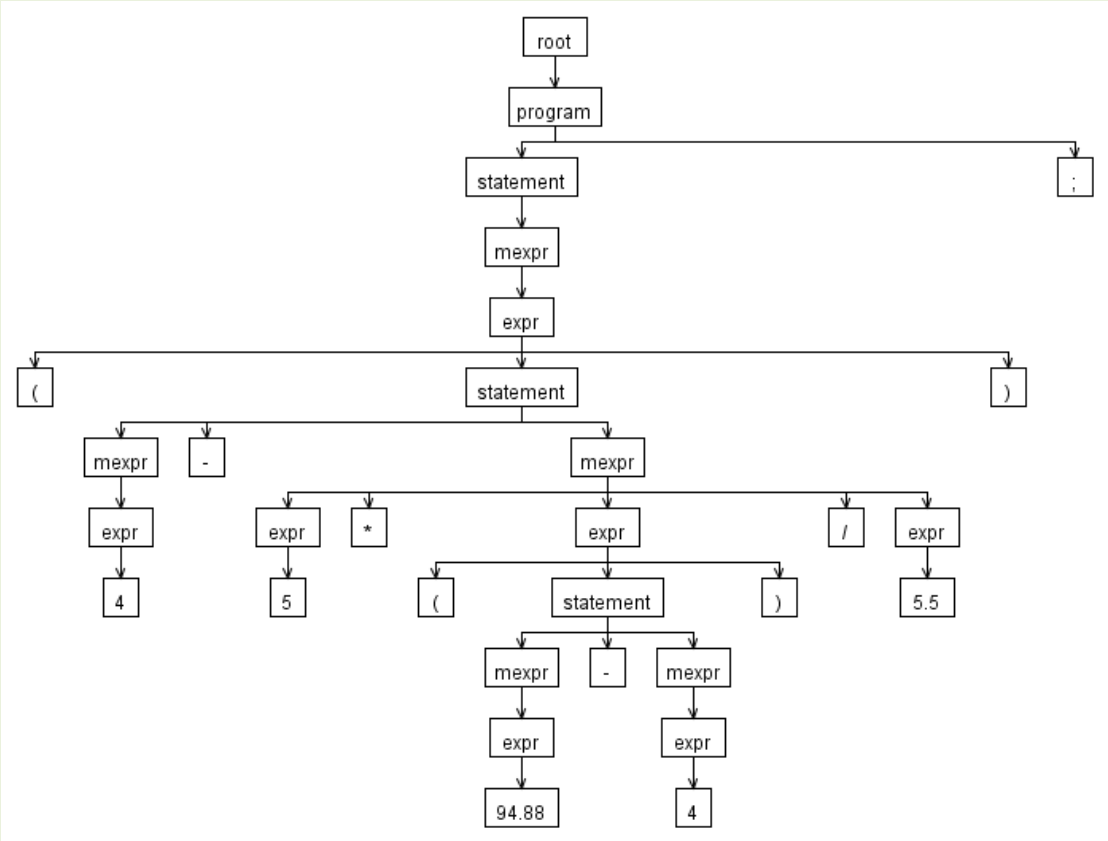
b.执行到“(4-5*(94.88-4 时：


ANTLRWorks 1.2.3
File Edit Find Go To Grammar Refactor Generate Debugger Window Help
F:\antlr\works\java\Calc\Calc.g
Calc.g
program
statement
mexpr
expr
ADD
SUB
MUL
DIV
LPAREN
RPAREN
INT
REAL
mexpr : expr ((MUL | DIV) expr)* ;
expr : INT | REAL | LPAREN statement RPAREN ;
ADD : '+' ;
Break on: All Location Consume LT Exception
Break on Consume
Input
(4-5*(94.88-4
Parse Tree
Stack
Rule
0 program
1 statement
2 statement
3 mexpr
4 expr
5 statement
6 mexpr
7 expr
8 mexpr
9 expr
Zoom
Input Output Parse Tree AST Stack Events
Syntax Diagram Interpreter Console Debugger
13 rules 19:2 Writable
16M of 66M

c.分析结束时：


ANTLRWorks 1.2.3
File Edit Find Go To Grammar Refactor Generate Debugger Window Help
F:\antlr\works\java\Calc\Calc.g
Calc.g
program
statement
mexpr
expr
ADD
SUB
MUL
DIV
LPAREN
RPAREN
INT
REAL
statement ';' ;
statement : mexpr ((ADD | SUB) mexpr)* ;
mexpr : expr ((MUL | DIV) expr)* ;
expr : INT | REAL | LPAREN statement RPAREN ;
Break on: All Location Consume LT Exception
Break on Terminate
Input
(4-5*(94.88-4)/5.5);
Parse Tree
Stack
Rule
root
program
statement
mexpr
expr
Zoom
Input Output Parse Tree AST Stack Events
Syntax Diagram Interpreter Console Debugger
13 rules 9:2 Writable
15M of 66M

其 parser tree 为：



对输入" (4-5*(94.88-4)/5.5);"该文法解析成功。

另外，不用 antlrworls IDE，可以新建一个 Calc.java 类，实现对输入的文法分析。

代码如下：

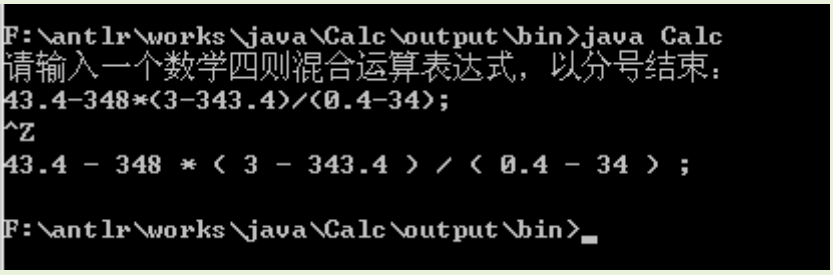
```
1  import org.antlr.runtime.*;
2  import org.antlr.runtime.tree.*;
3
4  public class Calc{
5
6    public static void main(String[] args)throws Exception {
7
8      System.out.println("请输入一个数学四则混合运算表达式，以分号结束：");
9
10     ANTLRInputStream input = new ANTLRInputStream(System.in);
11
12     CalcLexer lexer = new CalcLexer(input);
13     CommonTokenStream tokens = new CommonTokenStream(lexer);
14     CalcParser parser = new CalcParser(tokens);
15     CalcParser.program_return r = parser.program();
16     System.out.println( ((BaseTree)r.getTree()).toStringTree());
17
18   }
19 }
```

进入命令行后，用 javac 和 java 命令编译执行 Calc，再输入您想解析的表达式即可。

如输入"43.4-348*(3-343.4)/(0.4-34);"

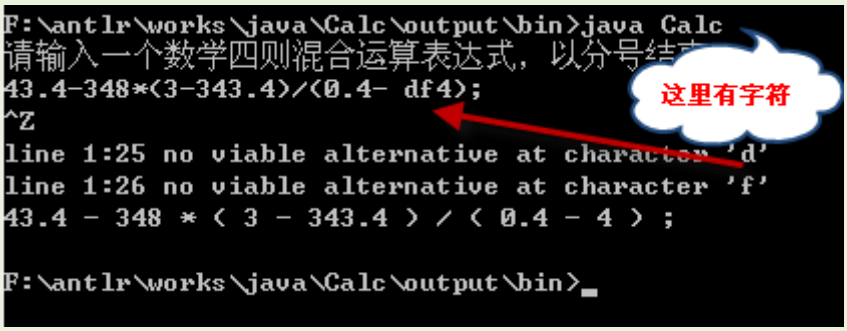
这时能够正确解析

为"43.4","-","348","*","(","3","-","343.4",")","/","(","0.4","-","34",")",";".如图所示：



当输入"43.4-348*(3-343.4)/(0.4-df4);"时，会出现解析错误提示，因为"df"在该文法中是未定义的。

此时，输出如图：



到此，Calc 简单实现就已经完成。