

# 编译原理读书笔记

---

学院： 计算机科学与技术学院

---

班级： **2014级 1班**

---

姓名： 孟维桥

---

学号： **3014216013**

---

内容：

---

学习笔记包括两部分，第一部分ANTLR书籍内容（放在了后面），第一部分遇到的问题及解决方法（放到了前面），书籍的阅读开始于5月5日，学习笔记的记录开始于5月6日。

书籍的阅读包括第1,2,3,4,5,6,10章，书籍的其他部分也进行了略读但未作记录。

## GitHub

---

ANTLR学习的相关笔记及相关代码托管于我的GitHub项目 [antlrgroup](#)

## 问题及解决

---

这一部分是最后添加的关于阅读过程中的一些问题的解决方法。

### 问题

---

1. 可以通过import导入lexer grammar,是否可以导入parser grammar，可否导入多个grammar?
2. section4.2中clear命令的实现

### 解决--见mycode

---

1. 一个g4文件可以声明为grammar、lexer grammar、parser grammar三种，统一通过import G1(G2,...)导入 (注：import这条命令只可以出现一次,有些死板...)，测试代码为mycode1/main.g4，编译运行测试如下：

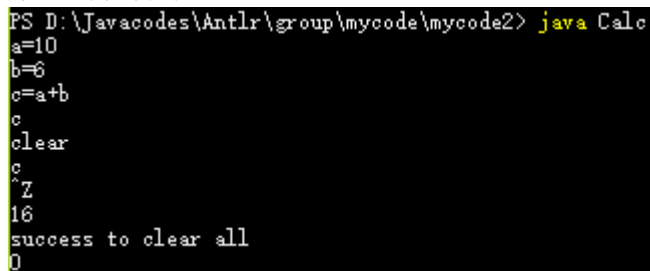
```
antlr4 main.g4
javac main*.java
grun main prog --gui
a=10
b=5
c=a+b
c
```

## 2. clear命令实现

- 方法一：通过chapter10中添加actions的方法（详见我们的工作myCalc），在此不再展示
- 方法二：在EvalVisitor.java中重载相关函数（代码mycode2）

```
antlr4 -no-listener -visitor LabeledExpr.g4
javac Calc.java LabeledExpr*.java
java Calc
a=10
b=6
c=a+b
c
clear
c
```

- 方法二测试结果：



```
PS D:\Javacodes\Antlr\group\mycode\mycode2> java Calc
a=10
b=6
c=a+b
c
clear
c
Z
16
success to clear all
0
```

## 我们的工作myCalc--见myCalc

- 用grammar actions的方法构建计算器
- 浮点数运算，实现了clear，阶乘，幂运算等
- 处理负数运算
- 测试运行

```
antlr4 -no-listener tools\Expr.g4
javac -d . tools\*.java
java tools.Calc
```

- 结果1：



```
PS D:\Javacodes\Antlr\group\mycode\myCalc> antlr4 -no-listener mytool2\Expr.g4
D:\Javacodes\Antlr\group\mycode\myCalc> java org.antlr.v4.Tool -no-listener mytool2\Expr.g4
PS D:\Javacodes\Antlr\group\mycode\myCalc> javac -d . mytool2\*.java
注：mytool2\Calc.java使用或覆盖了已过时的 API。
注：有关详细信息，请使用 -Xlint:deprecation 重新编译。
PS D:\Javacodes\Antlr\group\mycode\myCalc> java mytool2.Calc
3-5
-2.0
-3-5
-8.0
-4+5
1.0
a=-5
-a
5.0
-4*-5
20.0
```

- 结果2(乘方、clear等):

```
(-5)^2
25.0
a=(1-5)*(-6)-3
a
21.0
b=-a*a
b
-441.0
clear
success to clear all
a
0.0
b
0.0
```

# chapter1

---

## day1

---

学习使用Antlr第一天--Edit by Mwq-Mengweiqiao

今天初次安装并使用了Antlr，完成了第一章的学习，Antlr的学习之路开始了！

### About install:

1. Download <http://antlr.org/download/antlr-4.7-complete.jar>.
2. Add antlr-4.7-complete.jar to CLASSPATH(add The file .jar not the parent dir to CLASSPATH)
3. in the same fold,create dir /bin and Create batch commands for ANTLR Tool, TestRig

/bin/antlr4.bat:

```
java org.antlr.v4.Tool %*
```

/bin/grun.bat:

```
java org.antlr.v4.gui.TestRig %*
```

Then add /bin to PATH

### Have a try:

1. create a new file Hello.g4 and type into some code

```
grammar Hello;
r : 'hello' ID ;
ID : [a-z]+ ;
WS : [ \t\r\n]+ -> skip;
```

2. run antlr and compile:

```
antlr4 Hello.g4
javac *.java
```

3.test:

```
grun Hello r -tokens    #start TestRig on grammar Hello at rule r
#grun Hello r -tree     #show a tree
#grun Hello r -gui      #show gui
hello world             #input for the recognizer that you type
EOF                     #type Ctrl+Z on windows
```

## chapter2-The Big Picture

---

### day1--2.1 Let's Get Meta!

---

今天看了2.1节，本节介绍了本书的一些知识基础：

- 分析器parser--识别语言的程序
- 关于parse，（语法）分析parse分为两个阶段：词法分析阶段（词法分析器lexer)和（词法）分析阶段（真正的语法分析器parser)
- 语法分析树的生成

### day2--2.2 Implementing Parsers

---

章节2.2介绍的是Antlr工具根据语法规则所生成的递归下降分析器：

- 递归下降分析方法是自顶向下（自上而下）分析方法的一种实现方法
- 以迷宫的例子让读者更容易理解递归下降分析过程（虽然没什么必要）
- Antlr自动控制自上而下分析过程每个决定所需的lookahead的数量
- 引入了二义性的问题

### day3--2.3 You Can't Put Too Much Water into a Nuclear Reactor

---

这一节内容很有趣，尤其是这几个作者给出的二义性的句子（。。。写完之后突然发现这句话有歧义了），颠覆了在英语课所学到的not...too结构的认识。

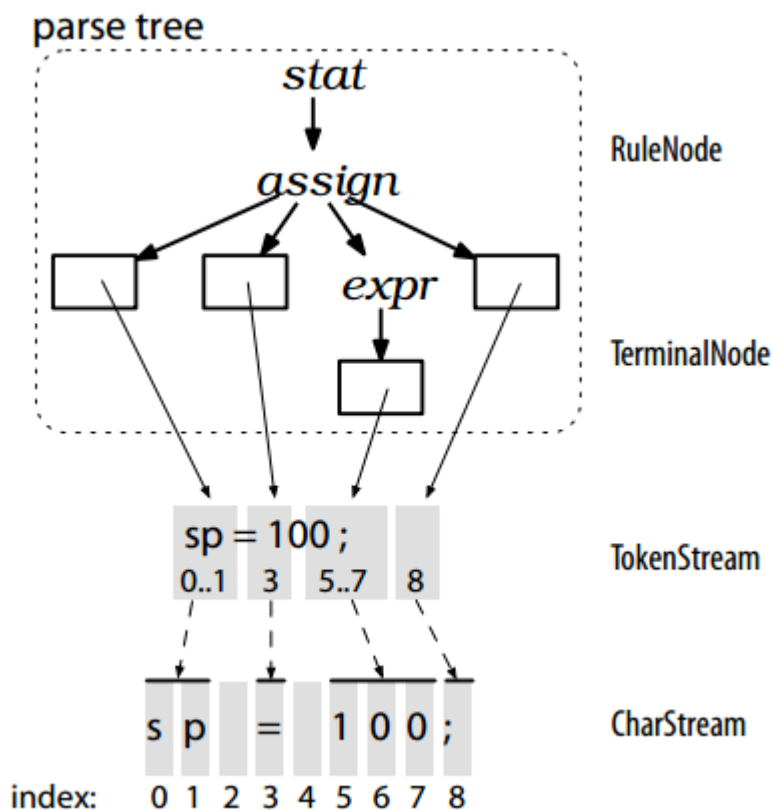
- For Whom No Thanks Is Too Much!竟然可以表示ungrateful,心疼那哥们的导师
- 本节主要对二义性问题进行介绍，展示了比较明显的二义性（如两个完全相同的可选项）和比较隐晦的二义性（通过f()的例子进行了详细分析）
- 二义性既出现于词法分析中，也出现在语法分析中，ANTLR对词法二义性的解决方法是采用最长匹配原则（通过begin与beginner的例子进行介绍）
- 语法二义性在现实中是无法避免的；C语言（以及其他带有指针\*的程序设计语言）引入了一种歧义（可以通过上下文分析解决）

### day4--2.4 Build Language Applications Using Parse Trees

---

本章节的主要内容是关于如何使用语法分析树来构建语言应用程序，相关要点如下：

- 构建语言应用程序需要对输入的短语执行适当的代码=>最简单的方法就是在语法分析器自动生成的语法分析树上进行操作=>回到了java领域（不必学习更为深入的ANTLR语法知识）
- 通过一张图示对语法分析过程中的要素与ANTLR中的java类之间的对应关系（包括CharStream, Lexer, Token, Parser, ParseTree等，以及连接词法分析和语法分析的管道--TokenStream）



- 对RuleNode和TerminalNode(ParseTree的子类)进行了详细介绍
- 介绍了context对象的概念和相关内容
- 通过对分析树的遍历，我们可以对树节点进行我们所需要的操作（包括计算结果、更新数据结构、生成输出）；然而我们不必每次去写重复的树遍历代码，我们可以使用ANTLR自动生成的树遍历机制

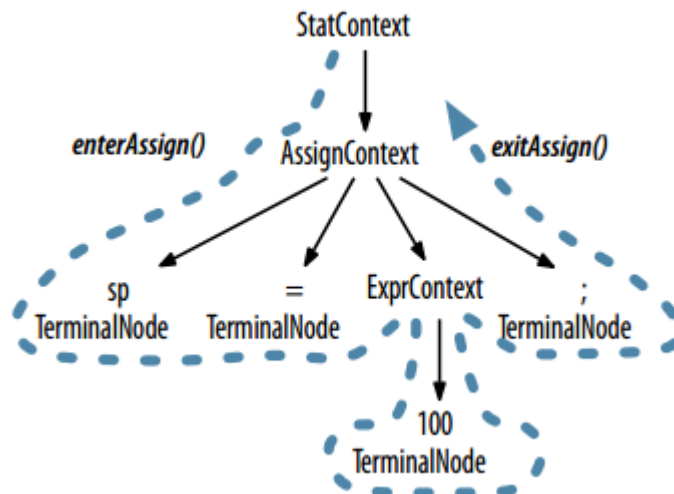
## day5--2.5 Parse-Tree Listeners And Visitors

ANTLR通过内置的遍历器walker提供了两种树遍历机制Listener和Visitor，在本节的最后对语法分析相关的术语概念进行了总结。

关于Listeners:

- Listener是基于回调机制的（对事件的处理，一般的java语言或者python语言程序采用的都是回调机制；QT开发平台对事件的处理声称采用的是信号-槽机制--至少我们用QT的时候看起来似乎是这样的）
- 每一条rule都有相对应的enter和exit方法（回调机制的基础）

- 图示1:



- 图示2:

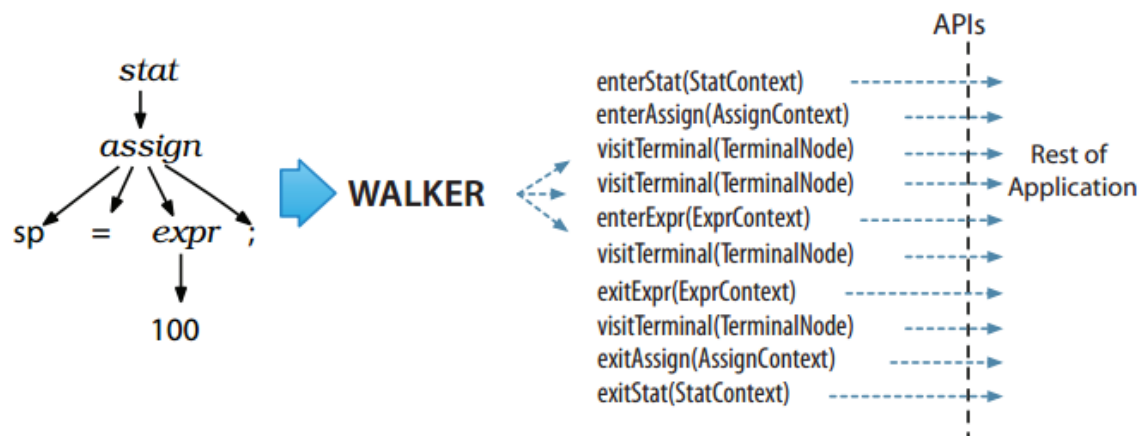
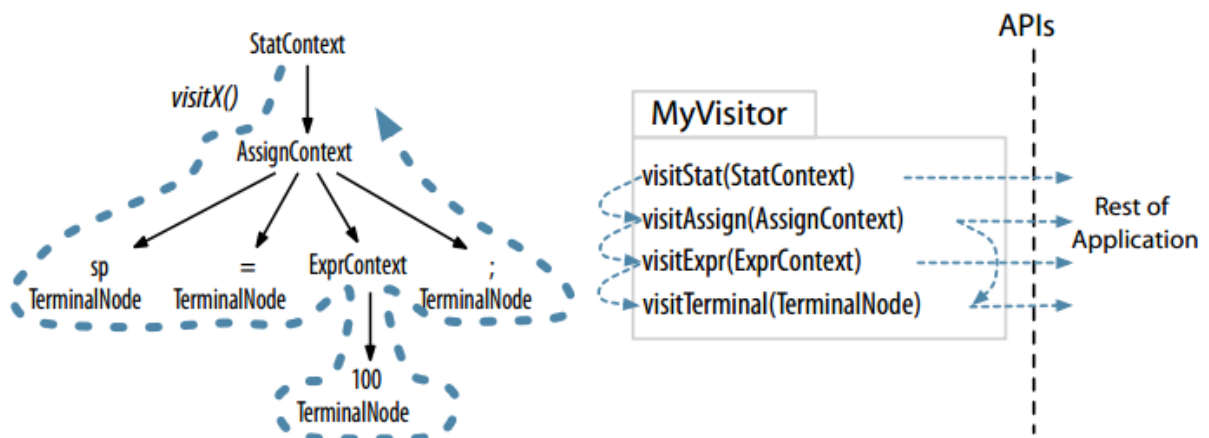


Figure 1—ParseTreeWalker call sequence

- Listener机制的美丽之处在于其是全自动的（我们不必去写分析树walker，同样listener也不必去明确地访问其子节点）

关于Visitors:

- Visitor用于访问操作指定的树节点（包括context node, terminalnode等），图示:



- 使用方法:

```
ParseTree tree=...; //tree is result of parsing
MyVisitor v=new MyVisitor();
v.visit(tree);
```

## chapter3

---

本章通过一个简单的ANTLR项目来学习ANTLR的基础用法：

- Java语言由.java得到.class文件时将short类型数组翻译为字符串来避免相关的限制
- 本章所要完成的项目就是简单的实现上述目的  
在本章末尾，对项目流程进行了总结。

### day1--3.1 The ANTLR Tool,Runtime,and Generated Code

---

本节对ANTLR的相关概念进行介绍，展示了语言应用构建的第一步--创建语法规则，并对之运行ANTLR：

- 通过ArrayInit.g4代码介绍了.g4代码的基本编写方法
- 运行ANTLR生成相关文件(.java,.tokens),并对相关文件作用进行介绍

在本节末，作者声称"ANTLR Grammars Are Stronger Than Regular Expressions",关于这一点，就先相信一下吧。

### day2--3.2 Testing the Generated Parser

---

本节没什么新内容，就是用-tokens模式、-tree模式和-gui模式测试了一下上节生成的语法分析器

### day3--3.3 Integrating a Generated Parser into a Java Program

---

本节讲述的内容是如何将生成的语法分析器整合到Java程序中，通过Test.java的例子进行详细介绍

### day4--3.4 Building a Language Application

---

本节讲述的是如何构建一个可实现数组翻译为字符串功能的语言应用程序：

- First, figure out how to convert each input token or phrase to an output string
- Then, code the translator,这意味着我们要通过重载一部分方法(与具体目的相关的一部分方法，如特定结点的enter、exit方法等)来编写继承了ArrayInitBaseListener接口类的回调函数类ShortToUnicodeString
- 编写语言应用程序Translate来实现预期目的（启动树遍历器walker,对分析树tree进行遍历，并以ShortToUnicodeString作为回调函数）

### day5--本章总结

---

构建语言应用程序基本流程为(以名称ArrayInit为例，相关细节见代码)：

- 编写ArrayInit.g4代码
- 运行ANTLR生成相关文件，（编译后）进行测试：

```
antlr4 ArrayInit.g4
javac *.java
grun ArrayInit init -tokens //init为开始规则,我们可以指定任意一条rule作为开始; 测试模式三种: -
token, -tree, -gui
{99,3,451}
EOF                                //Ctrl+Z on windows
```

- 编写回调函数类ShortToUnicodeString.java实现翻译功能（该类继承自ArrayInitBaseListener接口类）
- 编写应用程序类Translate.java,完成应用程序构建(并进行编译、调试、测试):

```
javac ArrayInit*.java Translate.java
java Translate
{99,3,451}
EOF
```

## chapter4

---

本章通过多个实例来展现ANTLR的能力所在，从而对ANTLR有一个快速的浏览，  
在1-4节通过四个主题的例子展示了ANTLR的特征，第5节是在词法分析层面对ANTLR的特征进行介绍

### day1--4.1 Matching an Arithmetic Expression Language

---

本节介绍的是算术表达式语言匹配的例子：

1. 通过可识别t.expr中类似算术表达式的Expr.g4代码的例子，向我们介绍了ANTLR语法规则的一些要点：

- Grammars由大量rules组成，分为语法规则rules和词法规则rules两种
- 语法规则rules以大写字母开头，词法规则以小写字母开头
- 多个可选项以"|"分开;其他如"+","\*","?"我们都很熟悉了
- WS rule中，"-> skip"要求词法分析器匹配并舍弃特定字符（如空格、回车、换行等）
- 输入串中每一个字符都必须成功匹配（但匹配之后可以舍弃）

2. ANTLR有处理（大多数种类的）左递归的能力

3. 通过ExprJoyRide.java的例子，展示了如何使用输入文件来获取待匹配串，并对ANTLR生成的相关文件进行测试使用

4. Importing Grammars--这种用法很常见也很常用：

- 可以将grammar分为lexer grammar和parser grammar作为模块"module"来管理，并通过import语句在需要的时候导入
- 使用时只需用antlr4命令及javac命令处理主代码，ANTLR会将代码中指定的模块自动导入

5. 错误处理机制：

- ANTLR能自动报告语法错误并从错误中恢复
- ANTLR的错误处理机制具有很大的灵活性，并且用户可以修改错误处理方法、捕获识别异常甚至修改底层的错误处理策略

### day2--4.2 Building a Calculator Using a Visitor

---



本节主要讲的是用Visitor来构建一个计算器，相关的变化有：

1. g4文件中用"# symbol"来对每一个可选项进行标注，以便ANTLR生成相关代码并在java代码中访问每一个可选项
2. 同时为操作符定义token名字，这样他们可以作为java常量来访问
3. 采用visitor模式（指明-no-listener -visitor)
4. 编写LabeledExprBaseVisitor的子类EvalVisitor，重载所需函数
5. 编译执行测试代码：

```
antlr4 -no-listener -visitor LabeledExpr.g4
javac Calc.java LabeledExpr*.java
java Calc t.expr
```

## day3--4.3 Building a Translator with a Listener

这一节主要讲的是用Listener来构建翻译器：

1. 如果想构建一个工具来从一个已定义的java类的方法生成java接口，可以想到的方法有java的反射机制（第一次接触到反射机制是有一次需要调用一个受到java保护的类，直接调用不被允许，后来发现了可以使用反射机制）和反编译工具，甚至可以尝试使用字节码库（如ASM。。。虽然这个完全没听说过）；而如果想要保留空格和注释，就now way了
2. Java.g4这份语法规则代码是作者参考oracle提供的java官方文档编写的（还好给了代码，不然。。。)
3. 对JavaBaseListener的部分方法进行重载，得到我们所需的ExtractInterfaceListener类
4. 编译执行测试代码：

```
antlr4 Java.g4
ls Java*.java
ls Extract*.java
javac Java*.java Extract*.java
java ExtractInterfaceTool Demo.java
```

## day4--4.4 Making Things Happen During the Parse

这一节主要讲的是如何通过在grammars中添加代码片段来将我们想要实现的actions添加到ANTLR自动生成的代码中，并借此

实现一个将输入数据的指定列输出的程序，在此之后介绍了通过语义谓词来动态控制一个语法规则的某些部分是否执行：

1. 在grammar中键入任意的actions
- 通过自定义构造函数来处理输入参数col
  - 通过if语句输出指定内容
  - 编译执行测试：

```
antlr4 -no-listener Rows.g4    #don't need the Listener
javac Rows*.java Col.java
java Col 1 < t.rows
java Col 2 < t.rows
java Col 3 < t.rows
```

## 2. 使用语义谓词修改语法分析器

- 语法谓词*{i <=n}?*--在条件成立( $i \leq n$ )时执行后续代码
- 编译执行测试:

```
antlr4 -no-listener Data.g4
javac Data*.java
grun Data file -tree t.data
```

# day5--4.5 Cool Lexical Features

这一节的内容有三部分:

## 1. 孤立语法规则: 处理同一文件中不同的格式

- ANTLR提供了lexical modes这样一种词法分析特性功能, 通过让词法分析器在遇到特定的标记字符的时候在不同的模式间进行切换, 从而达到更加方便的处理包含混合格式的文件的目的
- 以处理XML作为例子, 在遇到"<"和">"、"/>"时进行模式切换
- grun XML tokens中"tokens"指定以词法分析模式运行 (而非语法分析器模式)
- 编译执行测试:

```
antlr4 XMLLexer.g4
javac XML*.java
grun XML tokens -tokens t.xml
```

## 2. 改写输入流

- 改写输入流要做到的是在不影响插入点之外的一切的情况下在原始的token流之中插入特定的内容 (如常量域、特定代码行等)  
这对于源代码植入和重构问题而言是一种有效的解决策略
- 对于在java代码中插入serialization变量的例子, 只需要重载classbody的enter方法并编写listener代码即可
- 编译执行测试:

```
antlr4 Java.g4
javac InsertSerialID*.java Java*.java
java InsertSerialID Demo.java
```

## 3. 将tokens传递到不同的通道 (sending tokens on different channels)

- 通过词法分析器命令"->channel(HIDDEN)"可以将特定的内容 (如注释和空格等) 传递到HIDDEN通道, 语法分析器只处理默认通道 (忽略HIDDEN通道中的内容), 从而达到保留并忽视注释和空格的目的。
- 用法:

```
LINE_COMMENT
: '//' ~[\r\n]* '\r'? '\n' -> channel(HIDDEN)
;
```

## chapter5 Designing Grammars

1. 这一章对规则的设计进行了详细的介绍与说明，这一章包括6小节，分别介绍：从语言样例中引出语法规则、使用已存在的语法规则作为指导、用ANTLR规则识别一般的语言样式、处理过程左递归与结合性（一般为算符的左结合与右结合）、识别一般的词法结构、严格划分词法与语法。
2. 这一章我并没有像以前几章一样仔细的逐节逐段去读，而是选择了略读（包括后面的几章都是略读），后面的阅读中遇到问题再回头看这一章的内容。
3. 本章第三节介绍用ANTLR规则识别一般的语言样式，这一节详细介绍了ANTLR规则的写法与规范，是编译原理课程中词法语法规则在ANTLR中的具体写法。

## chapter6

这一章是前面章节中词法语法规则编写的一次实例演示，每一节提供了一种语言的简单的识别实现，没什么新的东西，就仔细的看了一下这几节的代码然后测试了一下这些例子

### 6.1 Parsing Comma-Separated Values

- 测试运行：

```
antlr4 CSV.g4
javac CSV*.java
grun CSV file -tokens data.csv
//redo with -gui and -tree
```

- 结果：

```
PS D:\Javacodes\Antlr\group\chapter6\section1> grun CSV file -tokens data.csv

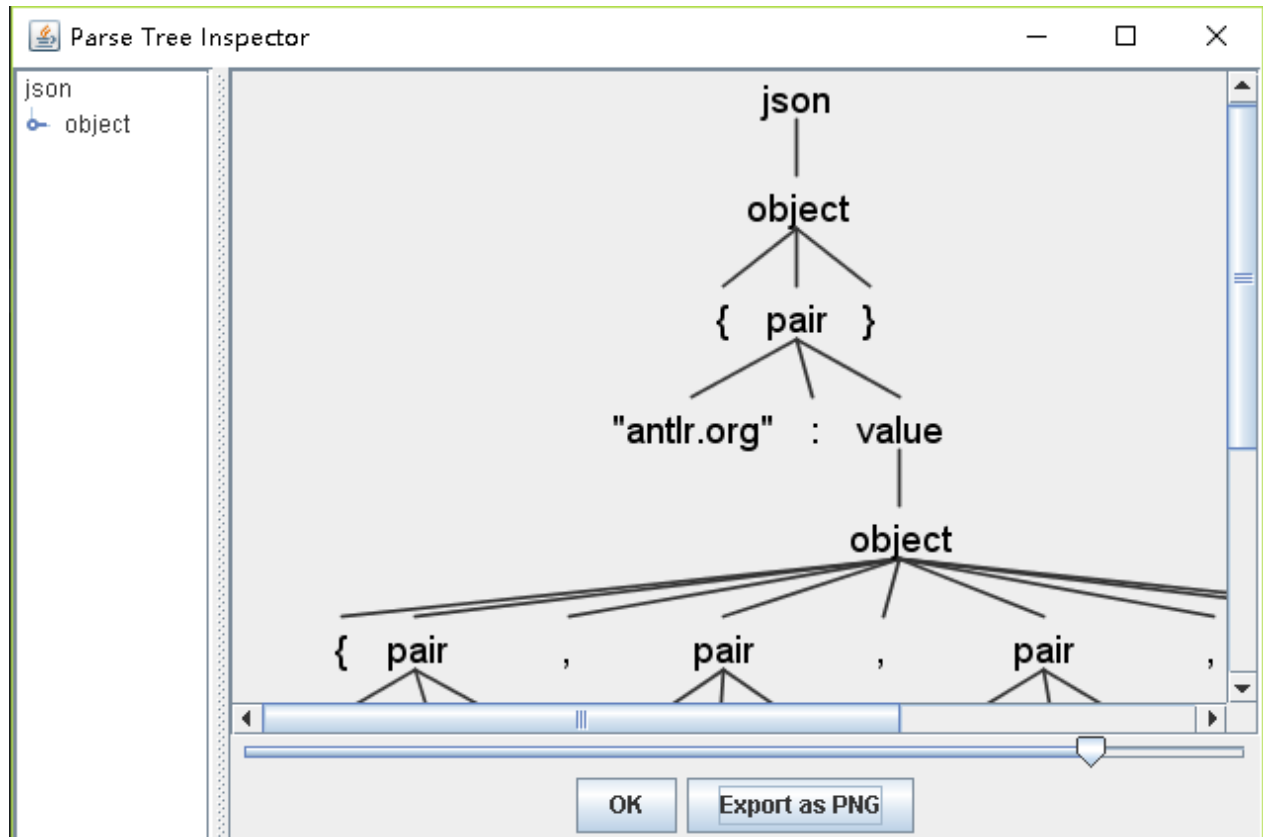
D:\Javacodes\Antlr\group\chapter6\section1> java org.antlr.v4.gui.TestRig CSV file -tokens data.csv
[0,0:6= 'Details', <TEXT>, 1:0]
[1,7:7= ',', <'>, 1:7]
[2,8:12= 'Month', <TEXT>, 1:8]
[3,13:13= ',', <'>, 1:13]
[4,14:19= 'Amount', <TEXT>, 1:14]
[5,20:20= '\n', <'>, 1:20]
[6,21:29= 'Mid Bonus', <TEXT>, 2:0]
[7,30:30= ',', <'>, 2:9]
[8,31:34= 'June', <TEXT>, 2:10]
[9,35:35= ',', <'>, 2:14]
[10,36:43= '$2,000', <STRING>, 2:15]
[11,44:44= '\n', <'>, 2:23]
[12,45:45= ',', <'>, 3:0]
[13,46:52= 'January', <TEXT>, 3:1]
[14,53:53= ',', <'>, 3:8]
[15,54:64= 'zippo', <STRING>, 3:9]
```

### 6.2 Parsing JSON

- 测试运行:

```
antlr4 JSON.g4
javac JSON*.java
grun JSON json -gui t.json
```

- 结果:

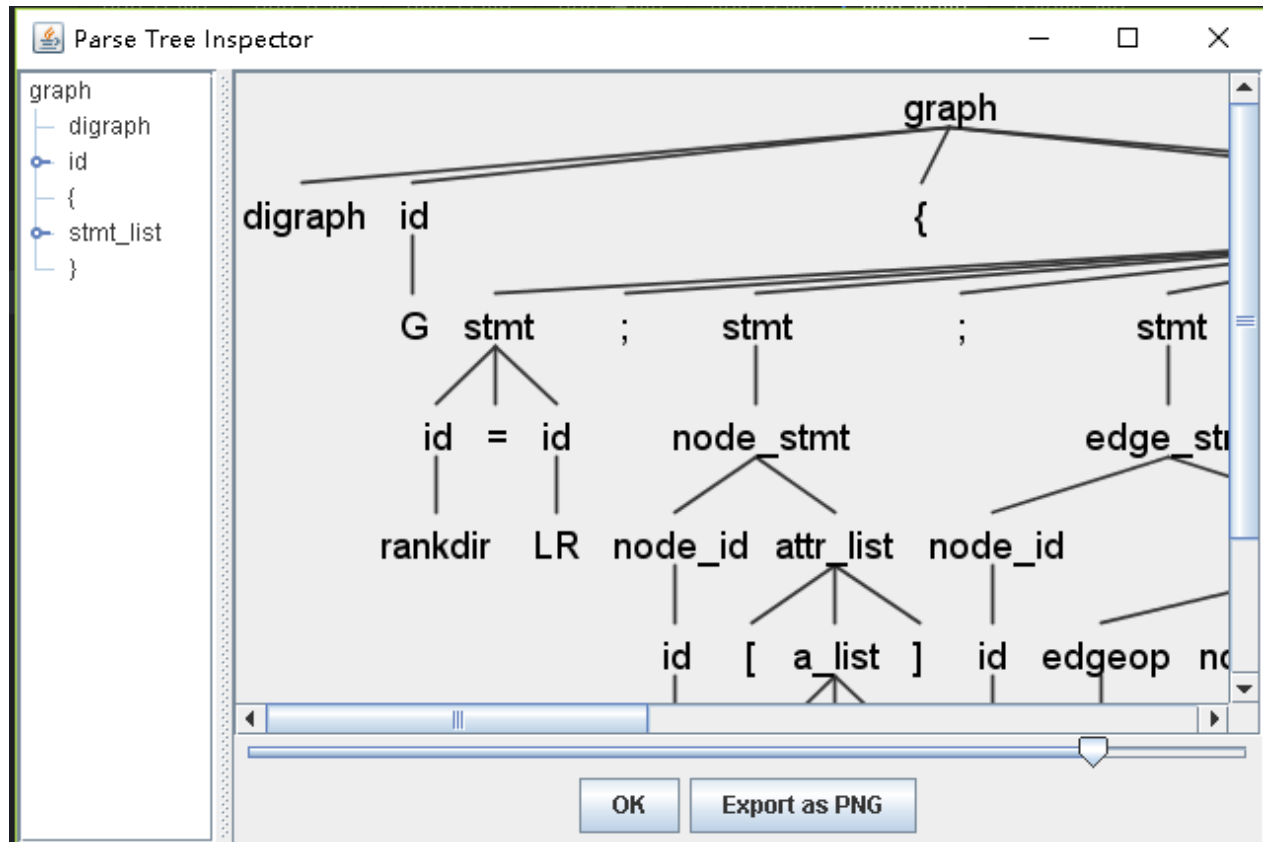


## 6.3 Parsing DOT

- 测试运行:

```
antlr4 DOT.g4
javac DOT*.java
grun DOT graph -gui t.dot
```

- 结果:

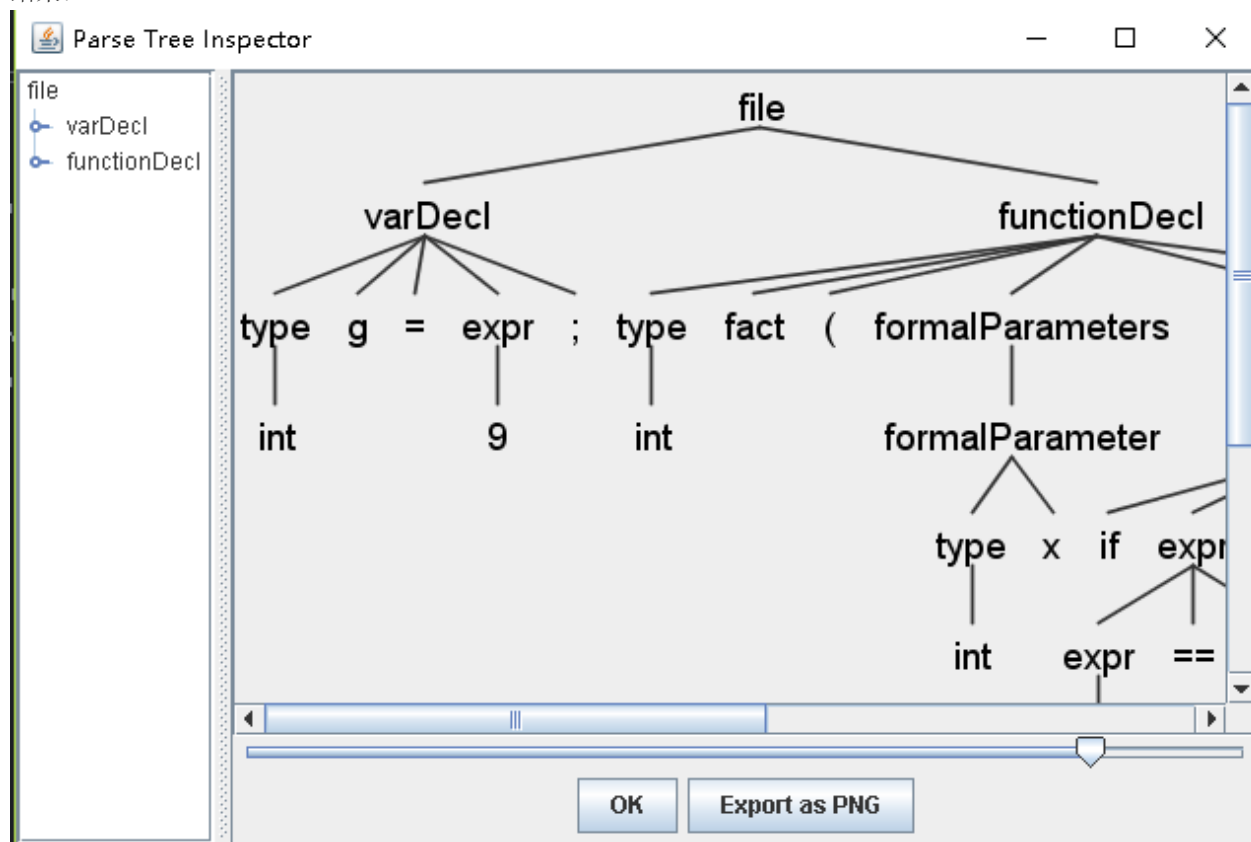


## 6.4 Parsing Cymbol

- 测试运行:

```
antlr4 Cymbol.g4
javac Cymbol*.java
grun Cymbol file -gui t.cymbol
```

- 结果:



## 6.5 Parsing R

- 测试运行:

```

antlr4 R.g4
javc R*.java
grun R prog -gui t.R
//javac TestR.java R*.java
//java TestR t.R

```

- 
- Parse Tree Inspector
- prog
- expr\_or\_assign
    - expr
      - addMe
      - expr
        - function
        - (
        - formlist
          - form
          - ,
          - form
        - )
        - expr
          - {
          - exprlist
          - }
    - <->
    - expr\_or\_assign
      - expr
        - addMe
        - sub
          - =
          - expr
          - exp
        - x
      - sublist
        - (
        - sublist
          - sub
            - x
            - =
            - expr
            - exp
          - ,
          - sub
        - )
  - \n
  - expr\_or\_assign
    - expr
      - addMe
      - expr
        - function
        - (
        - formlist
          - form
          - ,
          - form
        - )
        - expr
          - {
          - exprlist
          - }
    - \n
    - expr\_or\_assign
      - expr
        - addMe
        - expr
          - function
          - (
          - formlist
            - form
            - ,
            - form
          - )
          - expr
            - {
            - exprlist
            - }
      - <EOF>

OK Export as PNG

1. section6.5 tree.inspect找不到符号,该函数是在java代码中使用gui模式展示语法分析结果,但是在当前版本中该函数可能被替换为新的用法了,在网上搜了很多也查看了ANTLR的API文档,始终没有找到解决方法就先放弃了

我们的ANTLR项目就是基于本章的内容，通过Actions实现计算器的更多功能（阶乘、乘方、clear、浮点数、负数运算处理等），详细内容见项目报告。

1. 在刚刚之外使用动作。用注加。

- 指明package,语言,libraries的header部分

```
@header {
package mytool2;
import java.util.*;
import java.lang.*;
}
```

- 指明全局变量、常量、自定义函数的部分

```
@parser::members {
    /** "memory" for our calculator; variable/value pairs go here */
    Map<String, Double> memory = new HashMap<String, Double>();

    double eval(double left, int op, double right) {
        switch ( op ) {
            case MUL : return left * right;
            case DIV : return left / right;
            case ADD : return left + right;
            case SUB : return left - right;
            case POW : return Math.pow(left,right);
            case FAC : return vfac(left);
        }
        return 0;
    }
}
```

## 2. 将动作整合到规则中

- 基本用法:

```
stat:   e NEWLINE           {System.out.println($e.v);}
      | ID '=' e NEWLINE    {memory.put($ID.text, $e.v);}
      | NEWLINE
      ;
```

- 调用函数

```
e returns [double v]
: a=e op=('*' | '/' ) b=e  {$v = eval($a.v, $op.type, $b.v);}
| a=e op=('+' | '-' ) b=e  {$v = eval($a.v, $op.type, $b.v);}
| a=e op='^' b=e           {$v = eval($a.v, $op.type, $b.v);}
| a=e op='!'               {$v = eval($a.v, $op.type, 0.0); }
| number                   {$v = $number.v;}
| ID
{
    String id = $ID.text;
    $v = memory.containsKey(id) ? memory.get(id) : 0.0;
}
;
```

3. 动作Actions工作的原理是ANTLR在生成的java代码中自动创建相应的memory、函数等，在生成的代码中为每一条规则生成相应的上下文对象。



#### 4. 关于返回值的计算

- 对于integer值的获取，ANTLR默认提供了.int属性（ANTLR实现了visitInt函数来支持.int属性），代码如下：

```
@Override
public Integer visitInt(LabeledExprParser.IntContext ctx){
    return Integer.valueOf(ctx.INT().getText());
}
```

- 对于double值的获取，ANTLR没有提供默认实现，于是我自己写了visitDouble函数来提供对double值的获取，代码如下：

```
double visitDouble(String dtext){
    return Double.valueOf(dtext);
}
```

#### 5. 构建交互式的计算器

- 通过创建缓冲区将每一行的输入（以Enter作为结束）作为一条表达式

```
BufferedReader br = new BufferedReader(new InputStreamReader(is));
String expr = br.readLine();           // get first expression
int line = 1;                          // track input expr line numbers
```

- 对每一条表达式进行输入流处理、词法分析得到tokens流、语法分析产生结果(显示指定不创建语法语法分析树)
- 示例代码：

```
ExprParser parser = new ExprParser(null);
parser.setBuildParseTree(false);

while ( expr!=null ) {
    ANTLRInputStream input = new ANTLRInputStream(expr+"\n");
    ExprLexer lexer = new ExprLexer(input);
    lexer.setLine(line);
    lexer.setCharPositionInLine(0);
    CommonTokenStream tokens = new CommonTokenStream(lexer);
    parser.setInputStream(tokens);
    parser.stat();           // start the parser
    expr = br.readLine();    // see if there's another line
    line++;
}
}
```

## day2--10.2 Accessing Token and Rule Attributes

这一节提供了更多的Actions用法，包括局部变量locals[],重载初始化方法'@init{}',规则结束后的行为'@after{}'等，详细内容不再赘述。

