

# 天津大学

## 编译原理实践



组 长： 朱相举 (3019244319)

组 员： 李润泽 (3019244266)

遆铮 (3019244106)

吴俊杰 (3019207400)

高建鸿 (3019244316)

任课教师： 胡静老师

2022 年 9 月 17 日

# 第一章 初识 ANTLR

## 1) 安装 ANTLR 并且成功运行实例

```
命令提示符
Microsoft Windows [版本 10.0.19043.1706]
(c) Microsoft Corporation. 保留所有权利。

C:\Users\Zhao>antlr4

C:\Users\Zhao>java -cp D:\Prof_APPS\ANTLR\antlr-4.9.2-complete.jar;D:\Prof_APPS\ANTLR\antlr-4.9.2-complete.jar;D:\Java\jdk1.8.0_102\lib;D:\Prof_APPS\ANTLR;D:\Prof_APPS\ANTLR\grun.bat;D:\Prof_APPS\ANTLR\antlr4.bat; org.antlr.v4.Tool
ANTLR Parser Generator Version 4.9.2
  -o _____ specify output directory where all output is generated
  -lib _____ specify location of grammars, tokens files
  -atn _____ generate rule augmented transition network diagrams
  -encoding _____ specify grammar file encoding; e.g., euc-jp
  -message-format _____ specify output style for messages in antlr, gnu, vs2005
  -long-messages _____ show exception details when available for errors and warnings
  -listener _____ generate parse tree listener (default)
  -no-listener _____ don't generate parse tree listener
  -visitor _____ generate parse tree visitor
  -no-visitor _____ don't generate parse tree visitor (default)
  -package _____ specify a package/namespace for the generated code
  -depend _____ generate file dependencies
  -D<option>=value set/override a grammar-level option
  -Werror _____ treat warnings as errors
  -XdbgST _____ launch StringTemplate visualizer on generated code
  -XdbgSTWait _____ wait for STViz to close before continuing
  -Xforce-atn _____ use the ATN simulator for all predictions
  -Xlog _____ dump lots of logging info to antlr-timestamp.log
  -Xexact-output-dir all output goes into -o dir regardless of paths/package

C:\Users\Zhao>
```

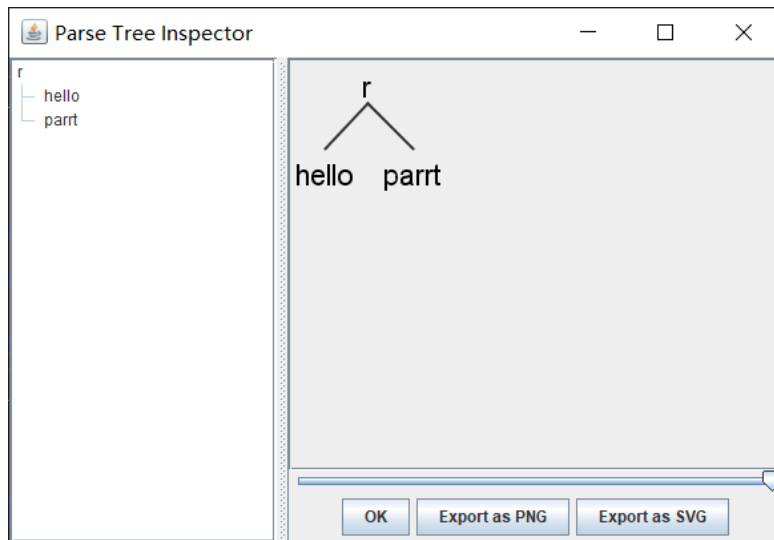
```
D:\Prof_APPS\ANTLR>antlr4 Hello.g4

D:\Prof_APPS\ANTLR>java -cp D:\Prof_APPS\ANTLR\antlr-4.9.2-complete.jar;D:\Prof_APPS\ANTLR\antlr-4.9.2-complete.jar;D:\Java\jdk1.8.0_102\lib;D:\Prof_APPS\ANTLR;D:\Prof_APPS\ANTLR\grun.bat;D:\Prof_APPS\ANTLR\antlr4.bat; org.antlr.v4.Tool Hello.g4
```

## 2) 运行 ANTLR 并测试识别程序

```
D:\Prof_APPS\ANTLR>grun Hello r -tokens

D:\Prof_APPS\ANTLR>java -cp .;D:\Prof_APPS\ANTLR\antlr-4.9.2-complete.jar;D:\Prof_APPS\ANTLR\antlr-4.9.2-complete.jar;D:\Java\jdk1.8.0_102\lib;D:\Prof_APPS\ANTLR;D:\Prof_APPS\ANTLR\grun.bat;D:\Prof_APPS\ANTLR\antlr4.bat; org.antlr.v4.gui.TestRig Hello r -tokens
hello parrrt
Z
[@0,0:4='hello',<'hello',1:0]
[@1,6:10='parrrt',<ID>,1:6]
[@2,13:12='<EOF>',<EOF>,2:0]
D:\Prof_APPS\ANTLR>
```



## 6.1 探索真实的语法世界

### 1) 认识 csv 格式文件

```
examples/data.csv
Details,Month,Amount
Mid Bonus,June,"$2,000"
,January,"""zippo""""
Total Bonuses,"","$5,000"
```

可以看到整个文档格式比较清晰明了，是标题行和常规行组成，其中我们可以将标题行提取出来，单独匹配，这样我们可以单独对他进行处理，如下

### 2) headrow

```
file : hdr row+ ;
hdr : row ;
```

同时，我们为了避免混淆，引入 `hdr`，意思是 `headrow`——标题行，以区别于一个常规的行，`row`。

至此，文档被分为 `headrow` 和普通的 `row`。

### 3) row 的规则

`row` 的规则可以简单的看出是——1 行由逗号分隔且由换号符终止的 `field`。

```
6 row : field (',' field)* '\r'? '\n' ;
```

为了让 field 定义更加灵活，我们允许两个逗号之间出现任意的文本、字符串，甚至没有。

## 4) field 的规则

```
8 field
9   :   TEXT    // 文本
10  |   STRING  // 字符串
11  |           // 空
12  ;
13
```

其中 TEXT 如注释所说，是文本，表示任意字符序列，除了什么字符呢？不能有逗号、换行\r\n，还有字符串的外框引号。

```
14 TEXT : ~[, \n\r"]+ ;
```

CSV 的规则规定除了字符序列之外，也可以用双引号包围，以使每个 field 内可以出现如逗号之类的字符，所以 csv 允许用双引号包围。通常我们使用两个双引号来转义。

## 5) 贪婪循环非贪婪循环

其中这里用到的 5.5 中的知识：

(nongreedy subrule) 的支持。非贪婪匹配的基本含义是：“获取一些字符，直到发现匹配后续子规则的字符为止”。更准确的描述是，在保证整个父规则完成匹配的前提下，非贪婪的子规则匹配数量最少的字符。有关非贪婪匹配的更多细节，请参阅15.6节。与之相反，.\*是贪婪的，因为它贪婪地消费掉一切匹配的字符（在本例中就是匹配通

配符的字符）。如果.\*? 令你感到迷惑不解，不要担心，只需要记住它是一种匹配双引号或者其他分界符之间的东西的模式即可。不久之后，我们在研究注释的章节中还会见到非贪婪循环。

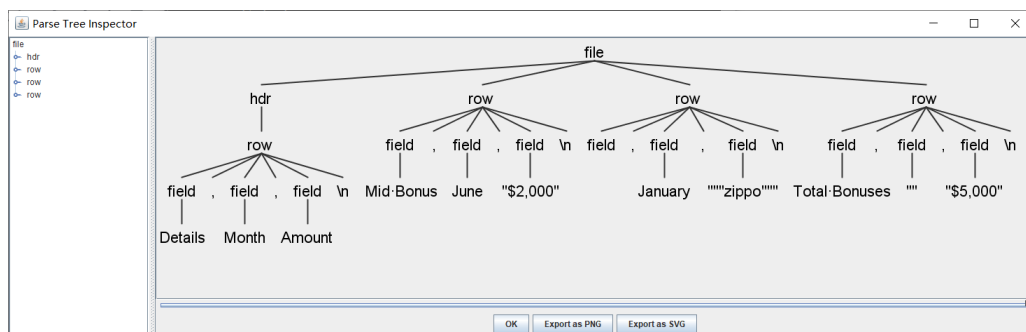
不能使用通配符构造非贪婪循环("""\.) \*?, 因为一旦遇到字符串开始之后第一个", 便会终止匹配过程。类似"x""y"会被匹配为两个字符串。

## 6) 测试词法符号流

```
9.2-complete.jar;D:\Prof_APPS\ANTLR\antlr-4.9.2-complete.jar;D:\Java\jdk1.8.0_102\lib;D:\Prof_APPS\ANTLR;D:\Prof_APPS\ANTLR\grun.bat;D:\Prof_APPS\ANTLR\antlr4.bat; org.antlr.v4.gui.TestRig CSV file -tokens t.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]
[@16,65:65='\n',<'>,3:20]
[@17,66:78='Total Bonuses',<TEXT>,4:0]
[@18,79:79=',',<'>,4:13]
[@19,80:81='"',<STRING>,4:14]
[@20,82:82=',',<'>,4:16]
[@21,83:90='"$5,000"',<STRING>,4:17]
[@22,91:91='\n',<'>,4:25]
[@23,92:91='<EOF>',<EOF>,5:0]
D:\2022大三下\项目制实践3\code\listeners>_
```

## 7) 语法分析树

```
9.2-complete.jar;D:\Prof_APPS\ANTLR\antlr-4.9.2-complete.jar;D:\Java\jdk1.8.0_102\lib;D:\Prof_APPS\ANTLR;D:\Prof_APPS\ANTLR\grun.bat;D:\Prof_APPS\ANTLR\antlr4.bat; org.antlr.v4.gui.TestRig CSV file -tree t.csv
(file (hdr (row (field Details) , (field Month) , (field Amount) \n)) (row (field Mid Bonus) , (field June) , (field "$2,000") \n) (row (field January) , (field ""zippo"" \n) (row (field Total Bonuses) , (field "" "$5,000") \n))
```



## 6.2 Parsing JSON

### json 文件

一个 JSON 文件可以是一个对象，或者是一个由若干个值组成的数组。从语法上看，这不过是一个选择模式，因此，我们可以用下列规则来表达：

```
json:  object
      |  array
      ;
```

### 对象

一个对象是一组无序的键值对集合；一个对象以一个左花括号（{）开始，并以一个（}）结束。每个键后跟一个冒号，键值对之间由逗号分隔。我们将键值对作为 `pair` 单独成一个文法规则。

```
object
:  '{ pair (',' pair)* }'
|  '{ }' // empty object
;
pair:  STRING ':' value ;
```

### 数组

数组是一组值的有序集合，一个数组由一个左方括号（[）开始，由一个右方括号（]）结束，其中的值由逗号分隔。数组包含一个由逗号分隔的序列模式和一个左右方括号间的语法符号依赖。

```
array
:  '[' value (',' value)* ']'
|  '[' ']' // empty array
;
```

### 值

Value 规则是一个选择模式，一个值可以是一个双引号包围的字符串，一个数字，true/false，null，一个对象，或者一个数组。这些结构可能发生嵌套。

value 规则使用字符串常量来匹配 JSON 中的关键字，因为我们把字符串作为一个整体，而不是一个字符序列看待。同理，数字也是相同。程序通常认为数字是完整的实体。

由于 value 规则引用了 object 和 array，它成为（间接）递归规则。通过 value 调用二者中的任一，最终都会回到 value 规则。

```
value
:   STRING
  |   NUMBER
  |   object // recursion
  |   array  // recursion
  |   'true'  // keywords
  |   'false'
  |   'null'
;
```

## 字符串

以上是所有的文法规则，接下来我们定义词法规则。

之前提过，我们不能匹配\和"，因此通过~[]的方式将它们除去。

此外转义字符的匹配我们单独成 ESC。

```

,

STRING :  '"' (ESC | ~["\\])* '"' ;
```

## 转义字符

以 fragment 开头的规则只能被其他的词法分析器规则使用，它们并不是词法符号

ESC 规则匹配一个 Unicode 序列或者预定义的转义字符。由于 ANTLR4 本身需要转义，因此我们匹配\之前还需要一个\。同理在[]中我们的\也需要在前面加一个\。

此外我们将\u+4 个 16 进制的匹配单独成 UNICODE。在 UNICODE 规则中，我们定义了一个 HEX 片段规则作为简写，来代替需要多次重复的十六进制数字。

```

fragment ESC :  '\\' ([\"\\bfnrt] | UNICODE) ;
fragment UNICODE : 'u' HEX HEX HEX HEX ;
fragment HEX : [0-9a-fA-F] ;

```

## 数字

JSON 语法参考中对数字的描述稍显复杂，我们可以将它整理成三个主要的备选分支。

编写数字之前我们最需要考虑的是，数字在 json 中最大的分类是实数。这意味着小数包含了指数，指数又包含了整数（笼统地讲），因此我们在写分支的时候需要按顺序写，使得匹配有正确的优先级。首先是小数 如 1.3e-9: 正负号（可选） 无符号整数 . 无符号整数 指数符号（可选）；其次是指数（无小数点）如 3e+9: 正负号（可选） 无符号整数 指数符号；最后是整数 如-1: 正负号（可选） 无符号整数。

```

NUMBER
:  '-'? INT '.' [0-9]+ EXP? // 1.35, 1.35E-9, 0.3, -4.5
|  '-'? INT EXP           // 1e10 -3e4
|  '-'? INT               // -3, 45
;

```

## 无符号整数&指数符号

根据 JSON 语法参考，我们知道，INT 不应当匹配除 0 之外的以 0 开头的数字。

我们在 NUMBER 规则中处理负号，这样，我们就能将精力集中于前两个备选分支：digit 和 digit1-9 digits。前者匹配任意的单个数字，所以 0 是可行的。后者匹配以 1 到 9，即除 0 之外开头的数字。

```

fragment INT : '0' | [1-9] [0-9]* ; // no leading zeros
fragment EXP : [Ee] [+-]? INT ; // \- since - means "range" inside [...]

```

## 空白字符

JSON 语法需要额外处理空白字符。

在任意两个词法符号之间，可以存在任意多的空白字符。



```
WS : [ \t\n\r]+ -> skip ;
```

现在，解析 JSON 的词法规则和语法规则都已就绪，我们可以将它们付诸实践了。首先，我们来打印输入文本[1,"u0049",1.3e9]中的词法符号。

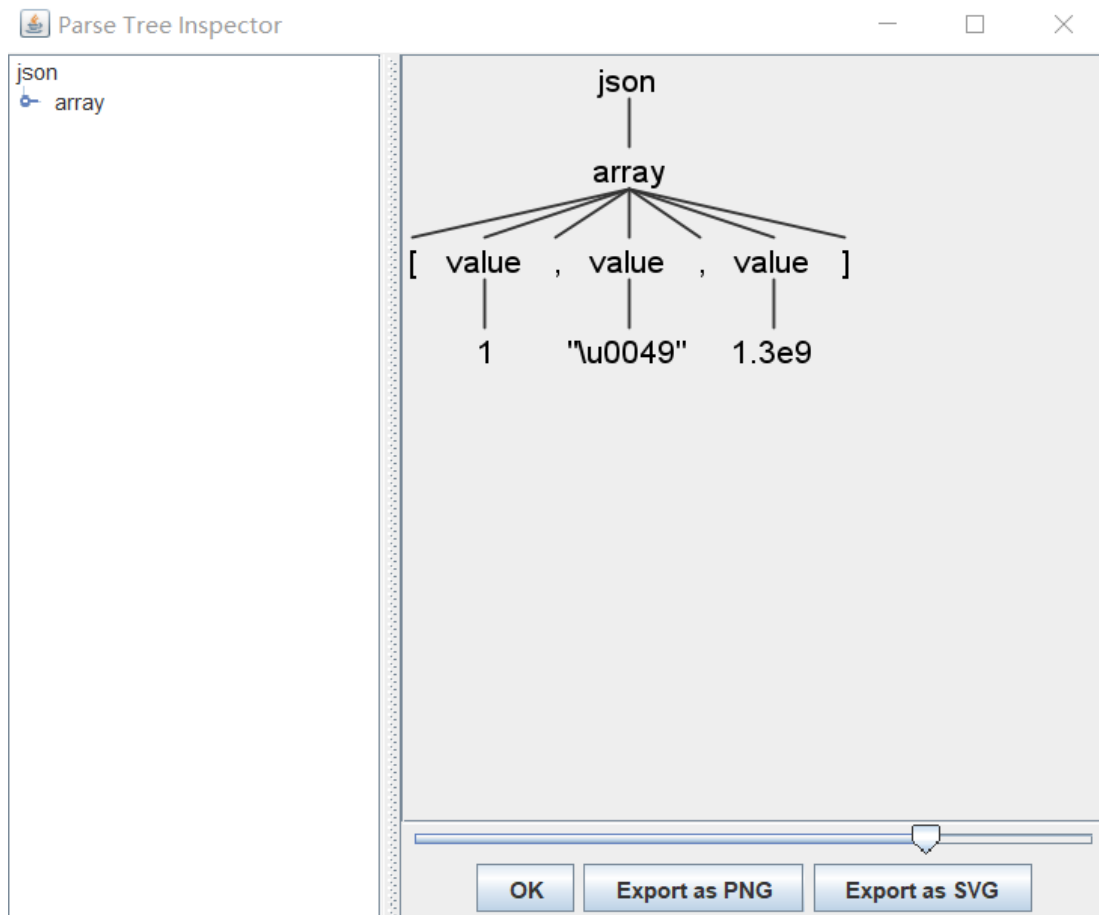
```
antlr4 -no-listener JSON.g4
```

```
javac JSON*.java
```

```
grun JSON json -gui
```

```
[1,"u0049",1.3e9]
```

```
ctrl+z
```



## 6.3 Parsing DOT

### 一、环境配置

#### 1. 下载 antlr4 包

```
curl -O http://www.antlr.org/download/antlr-4.0-complete.jar
```

在该目录下运行

```
C:\env\antlr>java -jar antlr-4.0-complete.jar
ANTLR Parser Generator Version 4.0
-o _____ specify output directory where all output is generated
-lib _____ specify location of grammars, tokens files
-atn _____ generate rule augmented transition network diagrams
-encoding _____ specify grammar file encoding; e.g., euc-jp
-message-format _____ specify output style for messages in antlr, gnu, vs2005
-long-messages _____ show exception details when available for errors and warnings
-listener _____ generate parse tree listener (default)
-no-listener _____ don't generate parse tree listener
-visitor _____ generate parse tree visitor
-no-visitor _____ don't generate parse tree visitor (default)
-package _____ specify a package/namespace for the generated code
-depend _____ generate file dependencies
-D<option>=value _____ set/override a grammar-level option
-Werror _____ treat warnings as errors
-XdbgST _____ launch StringTemplate visualizer on generated code
-XdbgSTWait _____ wait for STViz to close before continuing
-Xforce-atn _____ use the ATN simulator for all predictions
-Xlog _____ dump lots of logging info to antlr-timestamp.log
```

#### 2. 将该包加入 java 的 CLASSPATH 变量中

编辑系统变量

变量名(N):	CLASSPATH
变量值(V):	.;%JAVA_HOME%\tools.jar;%JAVA_HOME%\dt.jar;C:\env\antlr\antlr-4.0-complete.jar;

浏览目录(D)... 浏览文件(F)... 确定 取消


在其他目录测试运行

```
C:\Users\Lu shengcan>java org.antlr.v4.Tool
ANTLR Parser Generator Version 4.0
-o ____ specify output directory where all output is generated
-lib ____ specify location of grammars, tokens files
-atn generate rule augmented transition network diagrams
-encoding ____ specify grammar file encoding; e.g., euc-jp
-message-format ____ specify output style for messages in antlr, gnu, vs2005
-long-messages show exception details when available for errors and warnings
-listener generate parse tree listener (default)
-no-listener don't generate parse tree listener
-visitor generate parse tree visitor
-no-visitor don't generate parse tree visitor (default)
-package ____ specify a package/namespace for the generated code
-depend generate file dependencies
-D<option>=value set/override a grammar-level option
-Werror treat warnings as errors
-XdbgST launch StringTemplate visualizer on generated code
-XdbgSTWait wait for STViz to close before continuing
-Xforce-atn use the ATN simulator for all predictions
-Xlog dump lots of logging info to antlr-timestamp.log


C:\Users\Lu shengcan>_
```

### 3. 创建 bat 脚本

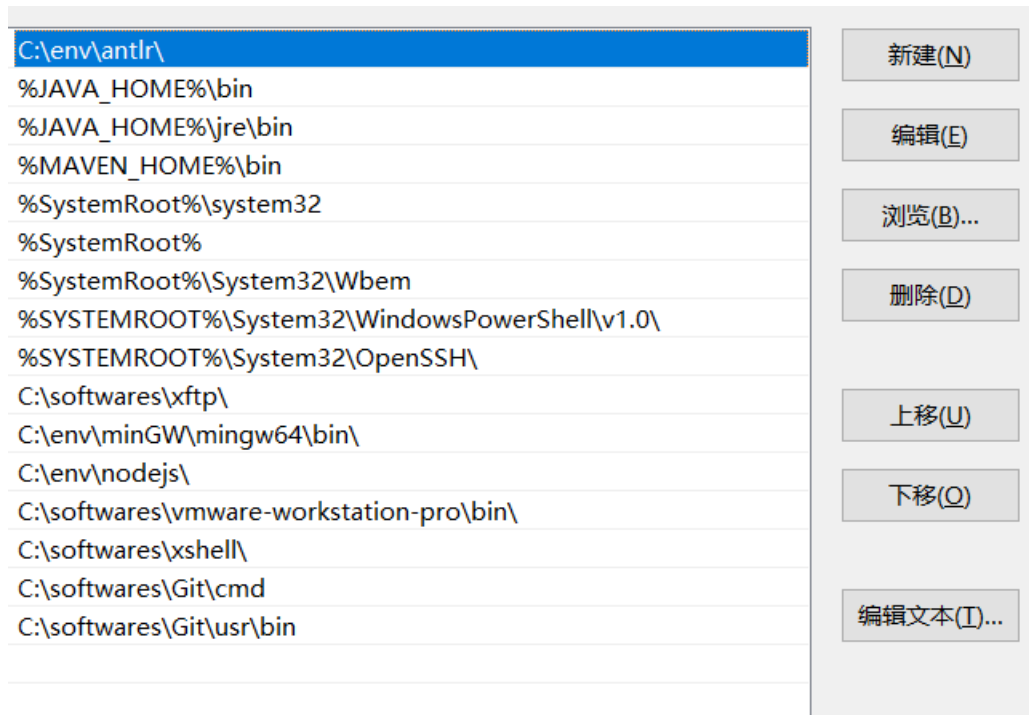
Antlr4.bat

```
C: > env > antlr >  antlr4.bat
1 java -cp C:\env\antlr\antlr-4.0-complete.jar;%CLASSPATH% org.antlr.v4.Tool %*
```

grun.bat

```
C: > env > antlr >  grun.bat
1 java -cp C:\env\antlr\antlr-4.0-complete.jar;%CLASSPATH% org.antlr.v4.runtime.misc.TestRig %*
```

### 3. 添加 bat 脚本至系统变量



### 4. 测试脚本

```
C:\Users\Lu shengcan>antlr4
C:\Users\Lu shengcan>java -cp C:\env\antlr\antlr-4.0-complete.jar;. ;C:\env\java-8\jdk1.8.0_311\lib\tools.jar; org.antlr.v4.Tool
ANTLR Parser Generator Version 4.0
  -o _____ specify output directory where all output is generated
  -lib _____ specify location of grammars, tokens files
  -atn _____ generate rule augmented transition network diagrams
  -encoding _____ specify grammar file encoding; e.g., euc-jp
  -message-format _____ specify output style for messages in antlr, gnu, vs2005
  -long-messages _____ show exception details when available for errors and warnings
  -listener _____ generate parse tree listener (default)
  -no-listener _____ don't generate parse tree listener
  -visitor _____ generate parse tree visitor
  -no-visitor _____ don't generate parse tree visitor (default)
  -package _____ specify a package/namespace for the generated code
  -depend _____ generate file dependencies
  -D<option>=value set/override a grammar-level option
  -Werror _____ treat warnings as errors
  -XdbgST _____ launch StringTemplate visualizer on generated code
  -XdbgSTWait _____ wait for STViz to close before continuing
  -Xforce-atn _____ use the ATN simulator for all predictions
  -Xlog _____ dump lots of logging info to antlr-timestamp.log
C:\Users\Lu shengcan>
```

```
C:\Users\Lu shengcan>grun
C:\Users\Lu shengcan>java -cp C:\env\antlr\antlr-4.0-complete.jar;. ;C:\env\java-8\jdk1.8.0_311\lib\tools.jar; org.antlr.v4.runtime.misc.TestRig
java org.antlr.v4.runtime.misc.TestRig GrammarName startRuleName
  [-tokens] [-tree] [-gui] [-ps file.ps] [-encoding encodingname]
  [-trace] [-diagnostics] [-SLL]
  [-input-filename(s)]
Use startRuleName='tokens' if GrammarName is a lexer grammar.
Omitting input-filename makes rig read from stdin.
```

## 二、解析 DOT 语言

### 1. 完成 DOT 语法

```
1 grammar DOT;
2
3 graph.....: STRICT? (GRAPH | DIGRAPH) id? '{' stmt_list '}' ;
4 stmt_list.....: ( stmt ';' ? ) * ;
5 stmt.....: node_stmt
6 |.....: edge_stmt
7 |.....: attr_stmt
8 |.....: id '=' id
9 |.....: subgraph
10 |.....: ;
11 attr_stmt.....: (GRAPH | NODE | EDGE) attr_list ;
12 attr_list.....: ( '[' a_list? ']' ) + ;
13 a_list.....: ( id ( '=' id ) ? ',' ? ) + ;
14 edge_stmt.....: ( node_id | subgraph ) edgeRHS attr_list ? ;
15 edgeRHS.....: ( edgeop ( node_id | subgraph ) ) + ;
16 edgeop.....: '->' | '--' ;
17 node_stmt.....: node_id attr_list ? ;
18 node_id.....: id port ? ;
19 port.....: ':' id ( ':' id ) ? ;
20 subgraph.....: ( SUBGRAPH id ? ) ? '{' stmt_list '}' ;
21 id.....: ID
22 |.....: STRING
23 |.....: HTML_STRING
24 |.....: NUMBER
25 |.....: ;
26
27 STRICT.....: [Ss][Tt][Rr][Ii][Cc][Tt] ;
28 GRAPH.....: [Gg][Rr][Aa][Pp][Hh] ;
29 DIGRAPH.....: [Dd][Ii][Gg][Rr][Aa][Pp][Hh] ;
30 NODE.....: [Nn][Oo][Dd][Ee] ;
31 EDGE.....: [Ee][Dd][Gg][Ee] ;
32 SUBGRAPH.....: [Ss][Uu][Bb][Gg][Rr][Aa][Pp][Hh] ;
33
34 NUMBER.....: '-' ? ( '.' DIGIT+ | DIGIT+ ( '.' DIGIT* ) ? ) ;
35 fragment
36 DIGIT.....: [0-9] ;
37
38 STRING.....: '"' ( '\\'' | . ) * ? '"' ;
39
40 ID.....: LETTER ( LETTER | DIGIT ) * ;
41 fragment
42 LETTER.....: [a-zA-Z\u0080-\u00FF_] ;
43
44 HTML_STRING.....: '<' ( TAG | ~[<>] ) * '>' ;
45 fragment
46 TAG.....: '<' .*? '>' ;
47
48 COMMENT.....: '/' * ? '*' / -> skip ;
49 LINE_COMMENT.....: '/' / ? '\r' ? '\n' -> skip ;
50
51 PREPROC.....: '#' .*? '\n' -> skip ;
52
53 WS.....: [ \t\n\r ] + -> skip ;
54
55
```

## 2. 生成 java 代码

```
学习资料\项目制3\antlr\DOT>antlr4 DOT.g4

学习资料\项目制3\antlr\DOT>java -cp C:\env\antlr\antlr-4.0-complete.jar;. ;C:\env\java
v\antlr\antlr-4.0-complete.jar; org.antlr.v4.Tool DOT.g4

学习资料\项目制3\antlr\DOT>
```

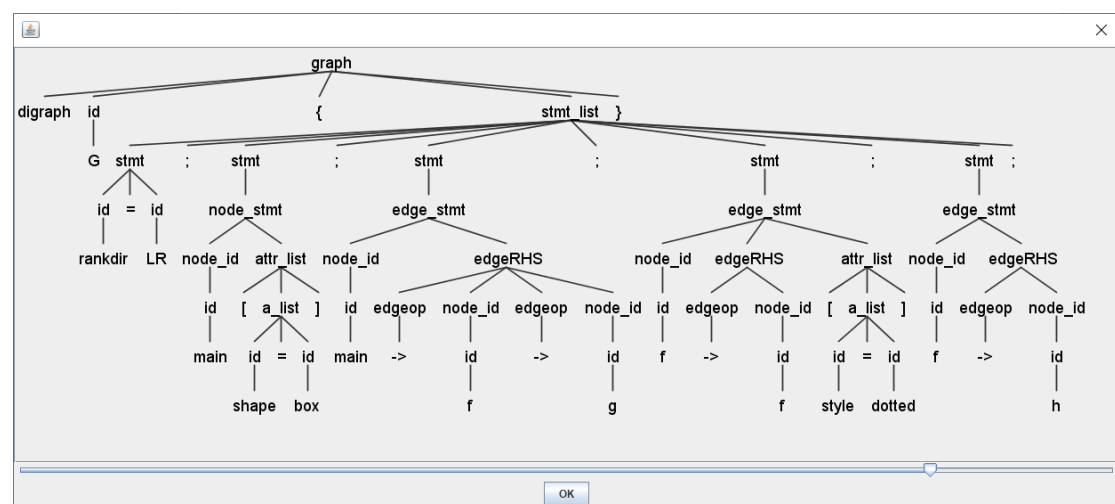
## 3. 编译 java 代码

```
学习资料\项目制3\antlr\DOT>javac DOT*.java
```

## 4. 生成 tokens

```
\\antlr\antlr-4.0-complete.jar; org.antlr.v4.runtime.misc.TestRig DOT_graph -tokens t.dot
@0,0:0= digraph', <13>, 1:0]
@1,8:8= G', <19>, 1:8]
@2,10:10= '[', <3>, 1:10]
@3,17:23= rankdir', <19>, 2:4]
@4,24:24= '=', <9>, 2:11]
@5,25:26= LR', <19>, 2:12]
@6,27:27= ',', <4>, 2:14]
@7,34:37= main', <19>, 3:4]
@8,39:39= '[', <3>, 3:9]
@9,40:44= shape', <19>, 3:10]
@10,45:45= '=', <9>, 3:15]
@11,46:48= box', <19>, 3:16]
@12,49:49= ']', <8>, 3:19]
@13,50:50= ',', <4>, 3:20]
@14,57:60= main', <19>, 4:4]
@15,62:63= ->', <1>, 4:9]
@16,65:65= f', <19>, 4:12]
@17,67:68= ->', <1>, 4:14]
@18,70:70= g', <19>, 4:17]
@19,71:71= ',', <4>, 4:18]
@20,78:78= f', <19>, 5:4]
@21,80:81= ->', <1>, 5:6]
@22,83:83= f', <19>, 5:9]
@23,85:85= '[', <3>, 5:11]
@24,86:90= style', <19>, 5:12]
@25,91:91= '=', <9>, 5:17]
@26,92:97= dotted', <19>, 5:18]
@27,98:98= ']', <8>, 5:24]
@28,99:99= ',', <4>, 5:25]
@29,106:106= f', <19>, 6:4]
@30,108:109= ->', <1>, 6:6]
@31,111:111= h', <19>, 6:9]
@32,112:112= ',', <4>, 6:10]
@33,115:115= ']', <10>, 7:0]
@34,118:117= EOF', <-1>, 8:0]
```

## 5. 生成语法分析树

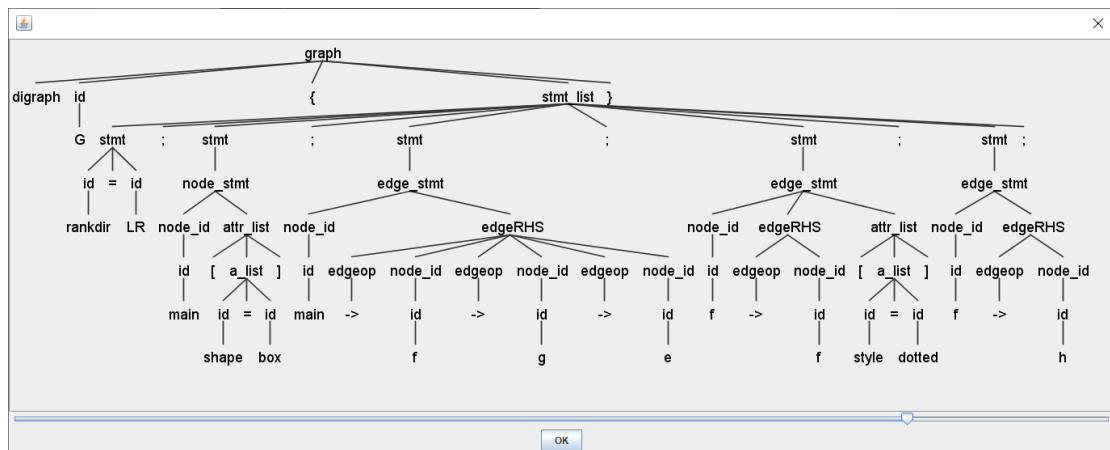


## 6. 其他两个测试用例

### 用例 1

```
DOT > 1.dot
1  digraph G {
2      ....rankdir=LR;
3      ....main [shape=box];
4      ....main -> f -> g -> e;
5      ....f -> f [style=dotted];
6      ....f -> h;
7  }
8
```

```
v:\antlr\antlr-4.0-complete.jar; org.antlr.v4.runtime.misc.TestRig DOT graph -tree 1.dot
(graph digraph (id G) { (stmt_list (stmt (id rankdir) = (id LR)) ; (stmt (node_stmt (node_id (id main)) (attr_list [ (a_list (id shape) = (id box)) ]))) ; (s
tmt (edge_stmt (node_id (id main)) (edgeRHS (edgeop ->) (node_id (id f)) (edgeop ->) (node_id (id g)) (edgeop ->) (node_id (id e)))))) ; (stmt (edge_stmt (nod
e_id (id f)) (edgeRHS (edgeop ->) (node_id (id f)) (attr_list [ (a_list (id style) = (id dotted)) ]))) ; (stmt (edge_stmt (node_id (id f)) (edgeRHS (edgeop
->) (node_id (id h)))))) ; } )
```



### 用例 2

```
DOT > 2.dot
1  digraph G {
2      ....rankdir=LR;
3      ....main [shape=box];
4      ....main -> f -> g -> e;
5      ....f -> f [style=dotted];
6      ....f -> h;
7      ....e -> g;
8  }
```





## 2.观察 Cymbol 程序

从最粗的粒度观察 Cymbol 程序,我们可以发现它由一系列全局变量和函数声明组成。

```
file: (functionDecl | varDecl)+ ;
```

同所有的类 c 语言一样,变量声明由一个类型开始,随后是一个标识符,最后是一个可选的初始化语句。

```
varDecl
    : type ID ('=' expr)? ';'
    ;
type: 'float' | 'int' | 'void' ; // user-defined types
```

## 3.函数声明

函数声明也基本上相同:类型后面跟着函数名,随后是被括号包围的参数列表,最后是函数体。

```
functionDecl
    : type ID '(' formalParameters? ')' block // "void f(int x) {...}"
    ;
formalParameters
    : formalParameter (',' formalParameter)*
    ;
formalParameter
    : type ID
    ;
```

一个函数体是由花括号包围的一组语句。

让我们先构造六种语句:嵌套的代码块、变量声明、if 语句、return 语句、赋值语句,以及函数调用。

## 4.ANTLR 语法表示

我们可以用下面的 ANTLR 语法来表达它们:

```

block: '{' stat* '}' ; // possibly empty statement block
stat: block
    | varDecl
    | 'if' expr 'then' stat ('else' stat)?
    | 'return' expr? ';'
    | expr '=' expr ';' // assignment
    | expr ';'          // func call
    ;

```

## 5.表达式语法

Cymbol 语言的最后一个主要部分是表达式语法。因为 Cymbol 实际上仅仅是其他语言的原型或者基础，因此没有必要包含非常多的运算符。假设我们的表达式包括一元取反、布尔非、乘法、加法、减法、函数调用、数组索引、等同性判断、变量、整数以及括号表达式。其中的重点是我们通常将备选分支按照从高到低的优先级进行排序。

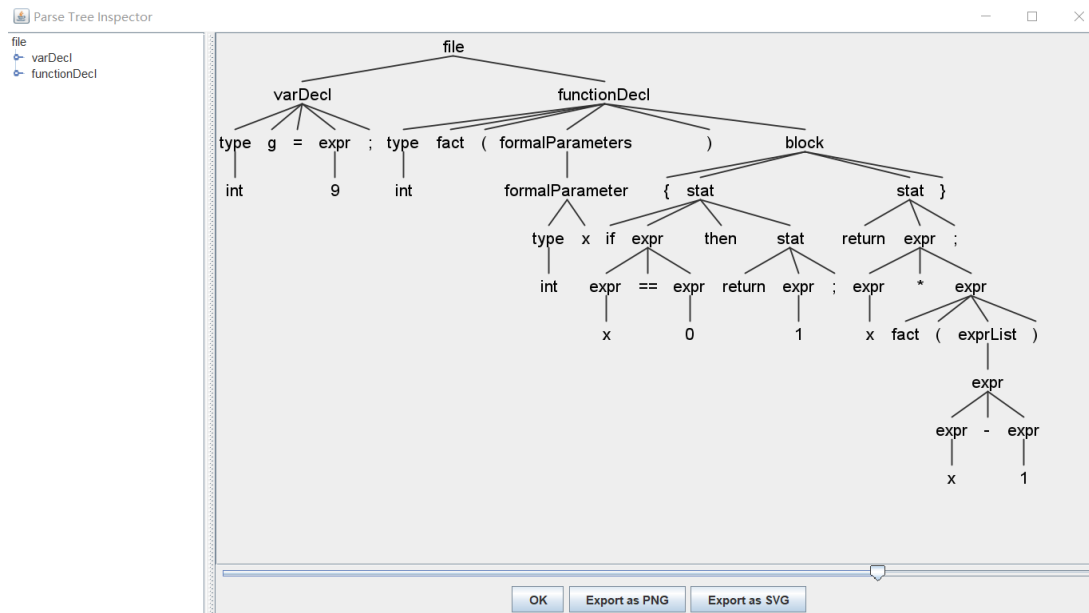
```

expr: ID '(' exprList? ')' // func call like f(), f(x), f(1,2)
    | ID '[' expr ']'      // array index like a[i], a[i][j]
    | '-' expr             // unary minus
    | '!' expr             // boolean not
    | expr '*' expr
    | expr ('+'|'-') expr
    | expr '==' expr      // equality comparison (lowest priority op)
    | ID                  // variable reference
    | INT
    | '(' expr ')'
    ;
exprList : expr (',' expr)* ; // arg list

```

## 6.实验测试

最后，输入相关 antlr 指令后输入指令：grun Cymbol file -gui t.cymbol。得到最终结果的语法树。



## 6.5 Parsing R

### (1) R 语言介绍

R 是一种用于描述统计问题的语言。例如可以很容易的创建向量，并应用函数并对其进行过滤。和其他语言一样，R 语言也是由一系列表达式以及函数组成。与其他语言不同的是，R 语言的赋值运算符有三个，分别是：`<=`，`=`，和 `<<=`，不过这个并不需要特别关心。

```

expr_or_assign
:  expr ('<-'|'|'<<-') expr_or_assign
|  expr
;

```

### (2) R 语言的语句

R 语言的语句分为种，陈述表达式，运算符表达式和函数表达式，用分号隔开，这与其他语言十分相似。我们可以通过如下进行匹配

```

| '{' exprlist '}' // compound statement
| 'if' '(' expr ')' expr
| 'if' '(' expr ')' expr 'else' expr
| 'for' '(' ID 'in' expr ')' expr
| 'while' '(' expr ')' expr
| 'repeat' expr
| '?' expr // get help on expr, usually string or ID
| 'next'
| 'break'
| '(' expr ')'

```

### (3) R 语言的运算符

R 语言存在许多运算符，这些运算符之间还存在着优先级关系，我们可以通过 R-lang 文档中的“中缀和前缀运算符”这一章来找到这些运算符的优先级关系。总而言之，我们可以使用

二进制、前缀和后缀运算符的方法来替代。

```

expr: expr '[' sublist ']' // '[' follows R's yacc grammar
| expr '[' sublist ']'
| expr ('::'|':::') expr
| expr ('$'|'@') expr
| expr '^'<assoc=right> expr
| expr ('-'|'+') expr
| expr ':' expr
| expr USER_OP expr // anything wrapped in %: '%' .* '%'
| expr ('*'|'/') expr
| expr ('+'|'-') expr
| expr ('>'|'>='|'<'|'<='|'=='|'!=') expr
| '!' expr
| expr ('&'|'&&') expr
| expr ('||'|'&&') expr
| '~' expr
| expr '~' expr
| expr ('->'|'>'|':') expr

```

## (4) R 语言的函数调用

R 语言通过 `formlist` 和 `sublist` 规则定义了形式参数定义列表和调用参数表达式。项目列表中以逗号分隔，每个项目可以是标识符，也可以是特殊标记。

我们可以使用类似于 `yacc` 中 `formlist` 的 ANTLR 规则对其进行编码。在 ANTLR 规则中每个参数可以是一个标记也可以是一个简单的表达式，也可为为空或者是特殊字段。

```
sublist : sub (',' sub)* ;
sub : expr
    | ID '='
    | ID '=' expr
    | STRING '='
    | STRING '=' expr
    | 'NULL' '='
    | 'NULL' '=' expr
    | '...'
    |
```

## (5) 标识符的匹配方式

根据 R-lang 的说法：标识符由字母、数字、句点和分数组成。它们不能以数字、下划线或句号开头按数字。

特殊的是...标识符可以以句点如'...'开头或者以'...1'，'...2'开头等等  
我们可以按照如下规则进行匹配。

```
ID : '.' (LETTER|'_'|'.') (LETTER|DIGIT|'_'|'.')*
    | LETTER (LETTER|DIGIT|'_'|'.')*
    ;
fragment LETTER : [a-zA-Z] ;
```

## (6) 语法分析树

