编译原理 ANTLR4

antlr学习笔记

基本使用

下载`antlr-4.7.1-complete.jar`，之后将其移入到java安装目录下的lib中，并添加环境变量

编写一个hello文法

```
grammar Hello; // 定义文法的名字
s : 'hello' ID ; // 匹配关键字hello，后面跟着一个标志符
ID : [a-z]+ ; // 匹配小写字母标志符
WS : [ \t\r\n]+ -> skip ; // 跳过空格、制表符、回车符和换行符
```

`java org.antlr.v4.Tool .\Hello.g` 生成代码

`javac *.java` 编译目录下的所有java文件

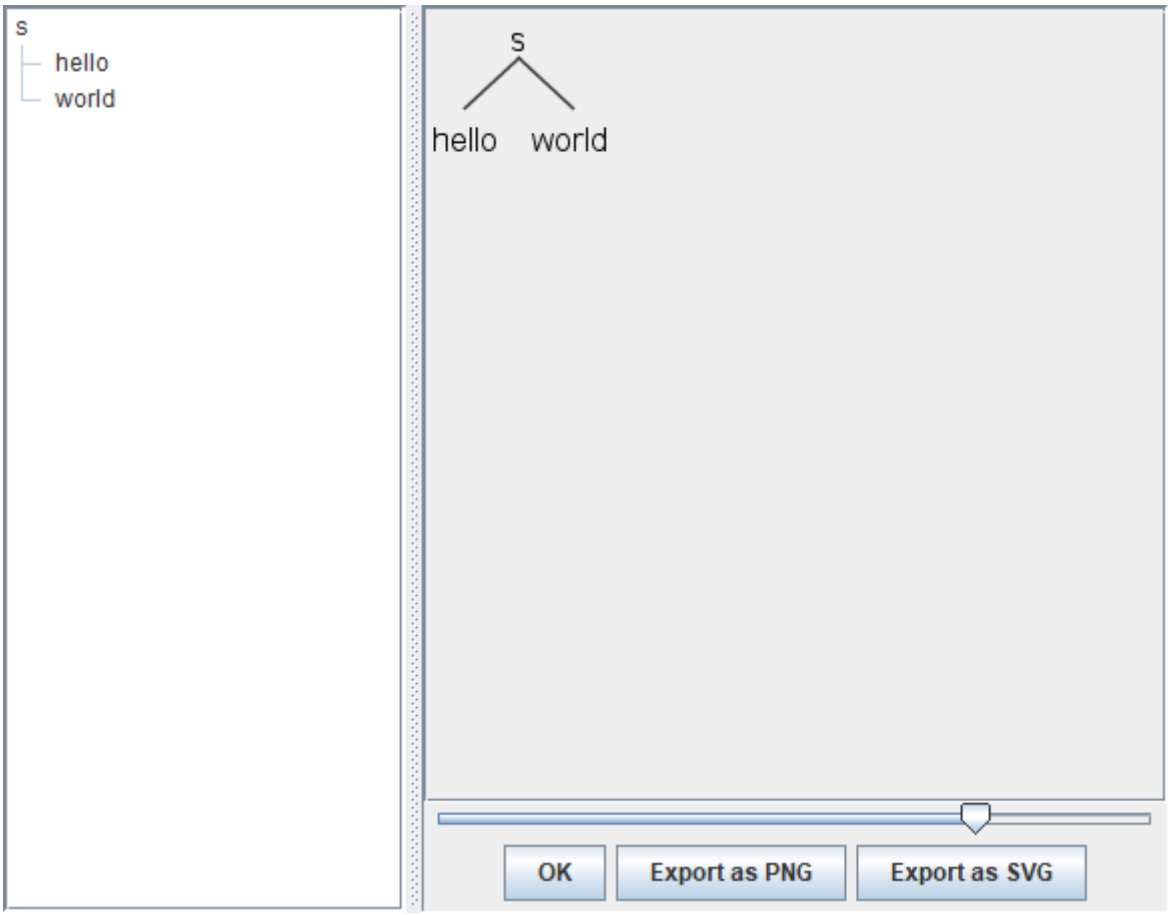
`java org.antlr.v4.gui.TestRig Hello s -tokens`

```
zz :: G:\2019\compile\bigwork\work0 » java org.antlr.v4.gui.TestRig Hello s -tokens
hello wrold
^Z
[@0,0:4='hello',<'hello'>,1:0]
[@1,6:10='wrold',<ID>,1:6]
[@2,13:12='<EOF>',<EOF>,2:0]

zz :: G:\2019\compile\bigwork\work0 »
```

`java org.antlr.v4.gui.TestRig Hello s -tree`

```
zz :: G:\2019\compile\bigwork\work0 » java org.antlr.v4.gui.TestRig Hello s -tree
hello world
^Z
(s hello world)
```

`java org.antlr.v4.gui.TestRig Hello s -gui`



基本概念

Listener 可以认为就是antlr自动帮你选择一条便利的路线
Visitor 可以自己规划路线，为了方便求值我后面都是用的visitor模式，同时antlr默认是Listener，所以需要添加`-visitor`

简单的计算器示例，除了原始的文法之外还添加了标签

```
grammar Calc;
prog
: stat+
;
stat
: expr # printExpr
```

```
    | ID '=' expr # assign
    ;
    expr
    : expr op=(MUL|DIV) expr # MulDiv
    | expr op=(ADD|SUB) expr # AddSub
    | INT # int
    | ID # id
    | '(' expr ')'  # parens
    ;
    MUL : '*' ;
    DIV : '/' ;
    ADD : '+' ;
    SUB : '-' ;
    ID : [a-zA-Z]+ ;
    INT : [0-9]+ ;
    WS : [ \t\r\n]+ -> skip ; // toss out whitespace
```

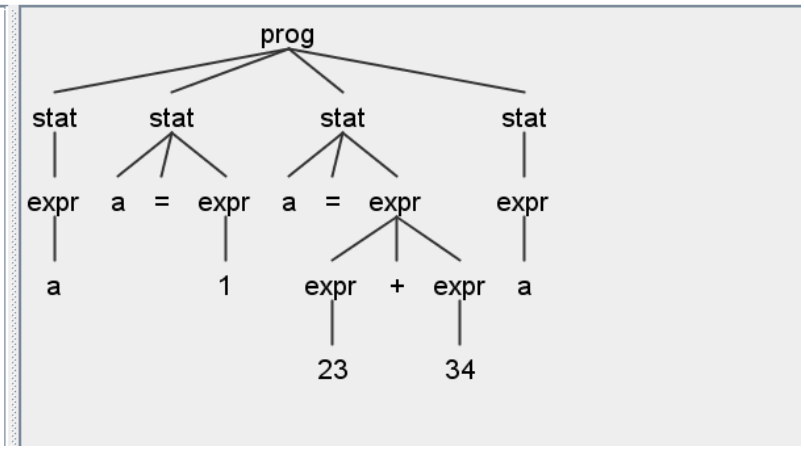
使用`-visitor`生成`CalcBaseVisitor.java`

然后编写`EvalVisitor.java`去继承该类，实现对应的方法

测试样例：

```
a
a=1
a=23+34
a
```

生成的语法



在IDEA中使用antlr

安装插件参照教程，主要是如何结合maven管理依赖

通过maven新建项目，引入antlr和antlr的maven插件依赖

```
        <dependency>
            <groupId>org.antlr</groupId>
            <artifactId>antlr4-runtime</artifactId>
            <version>4.5.1-1</version>
        </dependency>
        <dependency>
            <groupId>org.antlr</groupId>
            <artifactId>antlr4-maven-plugin</artifactId>
            <version>4.5</version>
            <scope>compile</scope>
        </dependency>
```

同时添加生成项目的配置：

- antlr添加`-visitor`选项
- 为了自动运行，添加了codehaus.mojo插件，指定compile.Main这个类运行

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.antlr</groupId>
                <artifactId>antlr4-maven-plugin</artifact
Id>
                <version>4.5</version>
                <configuration>
                    <arguments>
                        <argument>-visitor</argument>
                    </arguments>
                </configuration>
                <executions>
                    <execution>
                        <goals>
                            <goal>antlr4</goal>
```

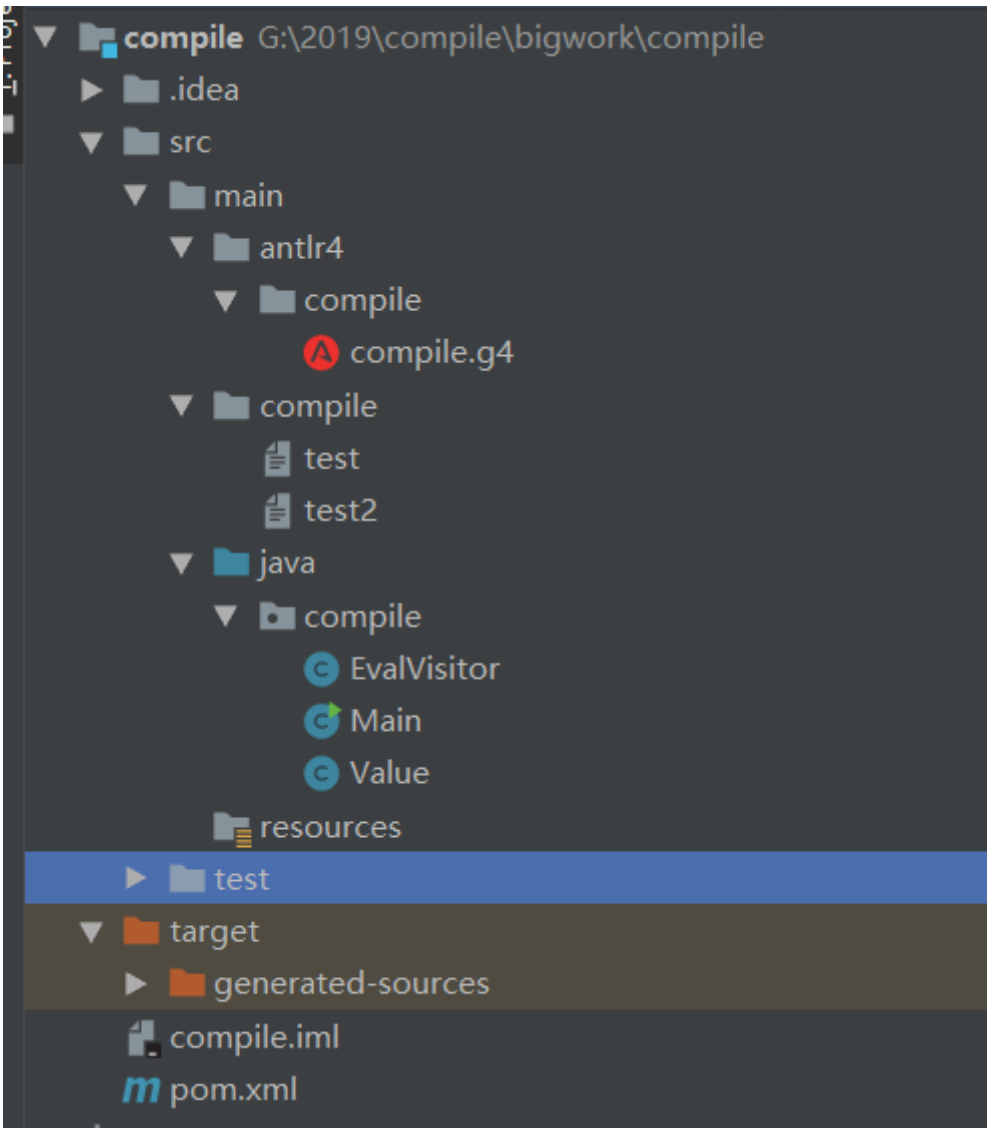
```
                        </goals>
                    </execution>
                </executions>
            </plugin>

            <plugin>
                <groupId>org.apache.maven.plugins</groupI
d>
                <artifactId>maven-compiler-plugin</artifa
ctId>
                <version>3.1</version>
                <configuration>
                    <source>1.7</source>
                    <target>1.7</target>
                </configuration>
            </plugin>

            <plugin>
                <groupId>org.codehaus.mojo</groupId>
                <artifactId>exec-maven-
plugin</artifactId>
                <version>1.2.1</version>
                <configuration>
                    <mainClass>compile.Main</mainClass>
                </configuration>
            </plugin>

        </plugins>
    </build>
```

项目结构：

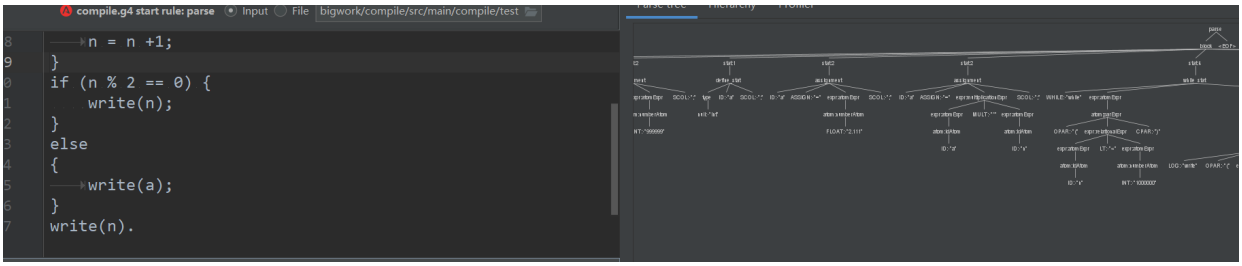
```
G:\2019\compile\bigwork\compile>tree
卷 Work 的文件夹 PATH 列表
卷序列号为 DE6B-5CA3
G:.
├─.idea
│  ├─dictionaries
│  └─inspectionProfiles
├─src
│  ├─main
│  │  ├─antlr4
│  │  │  └─compile
│  │  ├─compile
│  │  ├─java
│  │  │  └─compile
│  │  └─resources
│  └─test
│      └─java
└─target
    └─generated-sources
        └─antlr4
            └─compile
```

src下存放源码，`\src\main\antlr4`存放antlr的文法
`\src\main\java\compile`存放源代码
`\src\main\compile`存放测试样例

工程如下：



使用了antlr插件之后可以直接查看生成的语法树了



问题

最开始用的是直接下载的antlr-4.7.1，导致后续使用教程上的代码出错

修改为antlr.4.5.1就没事了

```
zz :: G:\2019\compile\bigwork\work4 » java org.antlr.v4.Tool -no-listener -visitor  .\Calc.g

zz :: G:\2019\compile\bigwork\work4 » cd .\out\production\work4\

zz :: G:\2019\compile\bigwork\work4\out\production\work4 » java Calc .\calc.txt
0
57
(prog (stat (expr a)) (stat a = (expr 1)) (stat a = (expr (expr 23) + (expr 34))) (stat (expr a)))

zz :: G:\2019\compile\bigwork\work4\out\production\work4 »
```

加标签的时候也遇到问题，一个表达式的所有分句要么全部加标签要么都不加