**语法结构**

语法本质上是一个语法声明，后跟一个规则列表，但是具有一般形式：

/\*\* Optional javadoc style comment \*/

grammar Name;

options {...}

import ... ;

tokens {...}

channels {...} // lexer only

@actionName {...}

rule1 // parser and lexer rules, possibly intermingled

...

ruleN

X必须调用包含语法的文件名X.g4。您可以按任何顺序指定options, imports, token, actions 。所有这些元素都是可选的，至少有一个规则。规则采用基本形式：

ruleName : alternative1 | ... | alternativeN ;

解析器规则名称必须以小写字母开头

词法分析器规则必须以大写字母开头。

要创建仅允许解析器规则的解析器语法，请使用以下标头。

parser grammar Name;

...

要创建仅允许词法分析器语法如下所示：

lexer grammar Name;

...

只有词法分析器语法才能包含mode规范。

只有词法分析器语法才能包含自定义通道规范

channels {

WHITESPACE\_CHANNEL,

COMMENTS\_CHANNEL

}

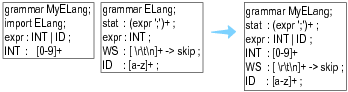
然后可以像词法规则中的枚举一样使用这些通道：

WS : [ \r\t\n]+ -> channel(WHITESPACE\_CHANNEL) ;

## Grammar Imports

import让你将语法分解成逻辑和可重用的块。语法从导入的语法继承所有规则，标记规范和命名操作。“主语法”中的规则覆盖导入的语法中的规则以实现继承。

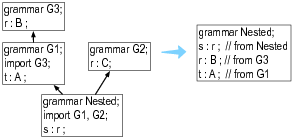
在下图中，右侧的语法说明了语法MyELang导入语法的效果ELang。

[](https://github.com/antlr/antlr4/blob/master/doc/images/combined.png)

MyELang继承规则stat，WS和ID，但覆盖规则expr和添加INT。这是一个示例构建和测试运行，显示MyELang可以识别整数表达式而原始表达式ELang不能。

如果有任何tokens规范，主语法将合并令牌集。如果有任何channel规范，主语法将合并通道集。任何命名的操作，例如@members将被合并。

导入的语法也可以导入其他语法。ANTLR以深度优先的方式遍历import的语法。如果两个或多个导入的语法定义规则r，ANTLR会选择r它找到的第一个版本。在下面的图中，ANTLR检查以下面的顺序的语法Nested，G1，G3，G2。

[](https://github.com/antlr/antlr4/blob/master/doc/images/nested.png)

并非每种语法都可以导入其他类型的语法：

* Lexer语法可以导入词法分析器，包括包含模式的词法分析器。
* 解析器可以导入解析器。
* 组合语法可以导入解析器或词法分析器。

ANTLR将导入的规则添加到主词法分析器语法的规则列表的末尾。

## Tokens Section

该tokens部分的目的是定义语法所需的令牌类型，其中没有关联的词法规则。基本语法是：

tokens { Token1, ..., TokenN }

大多数情况下，令牌部分用于定义语法中的操作所需的令牌类型

// explicitly define keyword token types to avoid implicit definition warnings

tokens { BEGIN, END, IF, THEN, WHILE }

@lexer::members { // keywords map used in lexer to assign token types

Map<String,Integer> keywords = new HashMap<String,Integer>() {{

put("begin", KeywordsParser.BEGIN);

put("end", KeywordsParser.END);

...

}};

}

该tokens部分实际上只定义了一组要添加到整个集合的标记。

$ cat Tok.g4

grammar Tok;

tokens { A, B, C }

a : X ;

$ antlr4 Tok.g4

warning(125): Tok.g4:3:4: implicit definition of token X in parser

$ cat Tok.tokens

A=1

B=2

C=3

X=4

**语法层面的行动**

header和members。前者在识别器类定义之前将代码注入生成的识别器类文件中，后者将代码作为字段和方法注入识别器类定义中。

这是一个语法为生成的代码指定包的示例：

grammar Count;

@header {

package foo;

}

@members {

int count = 0;

}

list

@after {System.out.println(count+" ints");}

: INT {count++;} (',' INT {count++;} )\*

;

INT : [0-9]+ ;

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

$ cd foo

$ antlr4 Count.g4 # generates code in the current directory (foo)

$ ls

Count.g4 CountLexer.java CountParser.java

Count.tokens CountLexer.tokens

CountBaseListener.java CountListener.java

$ javac \*.java

$ cd ..

$ grun foo.Count list

=> 9, 10, 11

=> EOF

<= 3 ints

Java编译器要求包中的类foo位于目录中foo。