Antlr首先收集词法，如下一个错误的示例：

Value : .\* ; // 匹配Value

WS : [\t\r\n\u0020]+ **->** skip ; // 跳过空格

.\*会匹配所有字符，所以空格也会被匹配

正确做法

Value : ~[\t\r\n\u0020]\* ;

WS : [\t\r\n\u0020]+ **->** skip ;

Lexer规则名称必须以大写字母开头：

TokenName : alternative1 | ... | alternativeN ;

您还可以定义帮助识别令牌。这些片段规则不会导致解析器可见的标记：

fragment

HelperTokenRule : alternative1 | ... | alternativeN ;

例如，DIGIT是一个非常常见的片段规则：

INT : DIGIT+ ;

fragment DIGIT : [0-9] ; // not a token by itself

**词汇模式**

除非指定mode命令，否则所有规则都被视为默认模式。组合语法中不允许使用模式，只有lexer语法

rules in default mode

...

mode MODE1;

rules in MODE1

...

mode MODEN;

rules in MODEN

...

**Lexer规则元素**

Lexer规则允许两个解析器规则不可用的结构：

|  |  |
| --- | --- |
| **句法** | **描述** |
| T | 将令牌T与当前输入位置匹配。 |
| ’literal’ | 匹配该字符或字符序列。例如，'while'或'='。 |
| [char set] | 匹配字符集中指定的一个字符。以下转义字符被解释为单个特殊字符：\ n，\ r，\ b，\ t，\ f，\ uXXXX和\ u {XXXXXX}。 |
| ’x’..’y’ | 匹配范围x和y之间的任何单个字符。例如，'a'..'z'。'a'..'z'与[az]相同。 |
| T | 调用lexer规则T;  ID：LETTER（LETTER |'0'..'9'）\*; |
| . | 点是一个匹配任何单个字符的单字符通配符。例：  退出 ： '\\' 。; //匹配任何转义的\ x字符 |
| {«action»} | end：（'endif'|'end'）{System.out.println（“找到了结束”）;}; |
| {«p»}? | 如果«p»在运行时评估为false，则周围规则变为“不可见”（不可行）。 |
| ~x | 匹配不在x描述的集合中的任何单个字符。〜（'x'|'y'|'z'）或〜[xyz]。 |

**递归Lexer规则**

ANTLR词法分析器规则可以是递归的。当你想匹配嵌套的动作块之类的嵌套标记时，这非常方便：{...{...}...}。

lexer grammar Recur;

ACTION : '{' ( ACTION | ~[{}] )\* '}' ;

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

**冗余字符串文字**

以下词法分析器语法定义了具有相同字符序列的两个标记：

lexer grammar L;

AND : '&' ;

mode STR;

MASK : '&' ;

解析器语法不能引用文字'＆'，但它可以引用标记的名称：

parser grammar P;

options { tokenVocab=L; }

a : '&' // error: no such token

AND // no problem

MASK // no problem

;

**Lexer命令**

词法分析器命令由->运算符后跟一个或多个可以选择接受参数的命令名组成：

TokenName : «alternative» -> command-name

TokenName : «alternative» -> command-name («identifier or integer»)

替代方案可以使用逗号分隔多个命令。以下是有效的命令名称：

* skip
* more
* popMode
* mode( x )
* pushMode( x )
* type( x )
* channel( x )

### skip

'skip'命令告诉词法分析器获取另一个令牌并丢弃当前文本。

ID : [a-zA-Z]+ ; // match identifiers

INT : [0-9]+ ; // match integers

NEWLINE:'\r'? '\n' ; // return newlines to parser (is end-statement signal)

WS : [ \t]+ -> skip ; // toss out whitespace

**mode（），pushMode（），popMode等**

模式命令改变模式堆栈，从而改变词法分析器的模式。'more'命令强制词法分析器获取另一个令牌但不丢弃当前文本。

// Default "mode": Everything OUTSIDE of a tag

COMMENT : '<!--' .\*? '-->' ;

CDATA : '<![CDATA[' .\*? ']]>' ;

OPEN : '<' -> pushMode(INSIDE) ;

...

XMLDeclOpen : '<?xml' S -> pushMode(INSIDE) ;

SPECIAL\_OPEN: '<?' Name -> more, pushMode(PROC\_INSTR) ;

// ----------------- Everything INSIDE of a tag ---------------------

mode INSIDE;

CLOSE : '>' -> popMode ;

SPECIAL\_CLOSE: '?>' -> popMode ; // close <?xml...?>

SLASH\_CLOSE : '/>' -> popMode ;

还可以看看：

lexer grammar Strings;

LQUOTE : '"' -> more, mode(STR) ;

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

mode STR;

STRING : '"' -> mode(DEFAULT\_MODE) ; // token we want parser to see

TEXT : . -> more ; // collect more text for string

### type()

lexer grammar SetType;

tokens { STRING }

DOUBLE : '"' .\*? '"' -> type(STRING) ;

SINGLE : '\'' .\*? '\'' -> type(STRING) ;

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

对于多个'type（）'命令，只有最右边的命令才有效。

### channel()

BLOCK\_COMMENT

: '/\*' .\*? '\*/' -> channel(HIDDEN)

;

LINE\_COMMENT

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

;

...

// ----------

// Whitespace

//

// Characters and character constructs that are of no import

// to the parser and are used to make the grammar easier to read

// for humans.

//

WS : [ \t\r\n\f]+ -> channel(HIDDEN) ;

从4.5开始，您还可以使用lexer规则上方的以下构造定义通道名称，例如枚举：

channels { WSCHANNEL, MYHIDDEN }