**解析器规则**

Java应用程序通过调用由ANTLR生成的规则函数来启动解析器。最基本的规则只是一个规则名称后跟一个以分号结尾的替代：

/\*\* Javadoc comment can precede rule \*/

retstat : 'return' expr ';' ;

规则也可以由|分隔

operator:

stat: retstat

| 'break' ';'

| 'continue' ';'

;

例如，这是一个带有空替代的规则，使整个规则可选：

superClass

: 'extends' ID

| // empty means other alternative(s) are optional

;

**替代标签**

我们可以通过使用＃运算符标记规则的最外层替代项来获得更精确的分析树监听器事件。

以下是带有标记替代品的两个规则。

grammar T;

stat: 'return' e ';' # Return

| 'break' ';' # Break

;

e : e '\*' e # Mult

| e '+' e # Add

| INT # Int

;

替代标签不必位于该行的末尾，并且＃符号后面不必有空格。ANTLR为每个标签生成规则上下文类定义。例如，这是ANTLR生成的侦听器：

public interface AListener extends ParseTreeListener

{

void enterReturn(AParser.ReturnContext ctx);

void exitReturn(AParser.ReturnContext ctx);

void enterBreak(AParser.BreakContext ctx);

void exitBreak(AParser.BreakContext ctx);

void enterMult(AParser.MultContext ctx);

void exitMult(AParser.MultContext ctx);

void enterAdd(AParser.AddContext ctx);

void exitAdd(AParser.AddContext ctx);

void enterInt(AParser.IntContext ctx);

void exitInt(AParser.IntContext ctx);

}

存在与每个标记的替代方案相关联的进入和退出方法。

如果替代名称与规则名称冲突，则ANTLR会出错。这是规则e的另一个重写，其中两个替代标签与规则名称冲突：

e : e '\*' e # e

| e '+' e # Stat

| INT # Int

;

从规则名称和标签生成的上下文对象获得大写，因此标记Stat与规则stat冲突：

$ antlr4 A.g4

错误（124）：A.g4：5：23：规则alt标签e与规则e冲突

错误（124）：A.g4：6：23：规则alt标签Stat与规则stat冲突

警告（125）：A.g4：2：13：令牌INT的隐式定义在解析器

**规则上下文对象**

ANTLR生成访问与每个规则引用关联的规则上下文对象（解析树节点）的方法。对于具有单个规则引用的规则，ANTLR生成不带参数的方法。请考虑以下规则。

inc : e '++' ;

ANTLR生成此上下文类：

public static class IncContext extends ParserRuleContext {

public EContext e() { ... } // return context object associated with e

...

}

当存在多个对规则的引用时，ANTLR还提供对访问上下文对象的支持：

field : e '.' e ;

ANTLR生成一个方法，该方法具有访问第i个元素的索引以及获取该规则的所有引用的上下文的方法：

public static class FieldContext extends ParserRuleContext {

public EContext e(int i) { ... } // get ith e context

public List<EContext> e() { ... } // return ALL e contexts

...

}

**规则元素标签**

您可以使用=运算符标记规则元素，以将字段添加到规则上下文对象：

stat: 'return' value=e ';' # Return

| 'break' ';' # Break

;

这里的value是规则e的返回值的标签。在这种情况下，标签值将成为ReturnContext中的一个字段，因为返回替代标签：

public static class ReturnContext extends StatContext {

public EContext value;

...

}

跟踪多个令牌通常很方便，您可以使用+ =“list label”运算符来执行此操作。例如，以下规则创建了与简单数组构造匹配的Token对象的列表：

array : '{' el+=INT (',' el+=INT)\* '}' ;

ANTLR在适当的规则上下文类中生成List字段：

public static class ArrayContext extends ParserRuleContext {

public List<Token> el = new ArrayList<Token>();

...

}

这些列表标签也适用于规则引用：

elist : exprs+=e (',' exprs+=e)\* ;

ANTLR生成一个包含上下文对象列表的字段：

public static class ElistContext extends ParserRuleContext {

public List<EContext> exprs = new ArrayList<EContext>();

...

}

**规则元素**

元素可以是规则，标记，字符串文字。这是规则元素的完整列表（稍后我们将详细介绍操作和谓词）：

|  |  |
| --- | --- |
| **句法** | **描述** |
| T | Token总是以大写字母开头。 |
| ’literal’ | 匹配当前输入位置的字符串文字。 |
| r | 解析器规则名称始终以小写字母开头。 |
| r [«args»] | 传入一个参数列表，就像函数调用一样。方括号内的参数位于目标语言的语法中，通常是以逗号分隔的表达式列表。 |
| {«action»} | 在前一个替代元素之后立即执行操作，紧接在下一个替代元素之前执行操作。该操作符合目标语言的语法。除了替换属性和令牌引用（例如$ x和$ xy）之外，ANTLR将动作代码复制到生成的类中。 |
| {«p»}? | 评估语义谓词«p»。如果«p»在运行时计算为false，则不要继续解析谓词。 |
| . | 匹配除文件标记结尾之外的任何单个标记。“点”运算符称为通配符。 |

如果要匹配除特定标记或标记集之外的所有内容，请使用~“not”运算符。

**子规则**

一个子规则就像一个缺少名称的规则，括在括号中。子规则在括号内可以有一个或多个备选项。子规则不能使用本地定义属性，而规则可以返回。有四种子规则（x，y和z代表语法片段）：

|  |  |
| --- | --- |
| **句法** | **描述** |
| [https://github.com/antlr/antlr4/raw/master/doc/images/xyz.png](https://github.com/antlr/antlr4/blob/master/doc/images/xyz.png) | （X | Y | Z）。匹配子规则中的任何替代方案。示例：  returnType : (type | 'void') ; |
| [https://github.com/antlr/antlr4/raw/master/doc/images/xyz_opt.png](https://github.com/antlr/antlr4/blob/master/doc/images/xyz_opt.png) | （X | Y | Z）？在子规则中不匹配或任何替代。示例：  classDeclaration : 'class' ID (typeParameters)? ('extends' type)? ('implements' typeList)? classBody ; |
| [https://github.com/antlr/antlr4/raw/master/doc/images/xyz_star.png](https://github.com/antlr/antlr4/blob/master/doc/images/xyz_star.png) | （x | y | z）\*匹配子规则中的替代零次或多次。示例：  annotationName : ID ('.' ID)\* ; |
| [https://github.com/antlr/antlr4/raw/master/doc/images/xyz_plus.png](https://github.com/antlr/antlr4/blob/master/doc/images/xyz_plus.png) | （x | y | z）+在子规则中匹配替代一次或多次。示例：  annotations : (annotation)+ ; |

**捕捉异常**

当规则中发生语法错误时，ANTLR捕获异常，报告错误，尝试恢复（可能通过消耗更多令牌），然后从规则返回。每条规则都包含在一个try/catch/finally声明中：

void r() throws RecognitionException

{

try {

rule-body

}

catch (RecognitionException re) {

\_errHandler.reportError(this, re);

\_errHandler.recover(this, re);

}

finally {

exitRule();

}

}

要更改单个规则的异常处理，请在规则定义后指定异常：

r : ...

;

catch[RecognitionException e] { throw e; }

这是一个完整的例外列表：

|  |  |
| --- | --- |
| **异常名称** | **描述** |
| RecognitionException | ANTLR生成的识别器抛出的所有异常的超类。它是RuntimeException的子类，以避免检查异常的麻烦。此异常记录了识别器（词法分析器或解析器）在输入中的位置，它在ATN中的位置（表示语法的内部图形数据结构），规则调用堆栈以及发生了什么类型的问题。 |
| NoViableAltException | 表示解析器无法通过查看剩余输入来决定要采用哪条路径。 |
| LexerNoViableAltException | 相当于NoViableAltException但仅适用于词法分析器。 |
| InputMismatchException时 | 当前输入令牌与解析器所期望的不匹配。 |
| FailedPredicateException | 在匹配令牌和调用规则的正常解析过程中，当语义谓词在预测之外评估为假时，解析器会抛出此谓词。 |

**规则属性定义**

rulename[args] returns [retvals] locals [localvars] : ... ;

这些[...]中定义的属性可以像任何其他变量一样使用。以下是将参数复制到返回值的示例规则：

// Return the argument plus the integer value of the INT token

add[int x] returns [int result] : '+=' INT {$result = $x + $INT.int;} ;

args，locals和return [...]通常是目标语言，但有一些约束。这些元素可以有初始化程序，[int x = 32, float y]

在任何参数，返回值或本地属性定义操作之后执行操作。row第10.2节“访问令牌和规则属性”中的规则前言很好地说明了语法：actions / CSV.g4

/\*\* Derived from rule "row : field (',' field)\* '\r'? '\n' ;" \*/

row[String[] columns]

returns [Map<String,String> values]

locals [int col=0]

@init {

$values = new HashMap<String,String>();

}

@after {

if ($values!=null && $values.size()>0) {

System.out.println("values = "+$values);

}

}

: ...

;

Rule行接受参数列，返回值，并定义局部变量col。方括号中的“动作”将直接复制到生成的代码中：

public class CSVParser extends Parser {

...

public static class RowContext extends ParserRuleContext {

public String [] columns;

public Map<String,String> values;

public int col=0;

...

}

...

}

生成的规则函数还将规则参数指定为函数参数，但它们会快速复制到本地RowContext对象中：

public class CSVParser extends Parser {

...

public final RowContext row(String [] columns) throws RecognitionException

{

RowContext \_localctx = new RowContext(\_ctx, 4, columns);

enterRule(\_localctx, RULE\_row);

...

}

...

}

使用逗号分隔同一操作中的属性：

a[Map<String,String> x, int y] : ... ;

ANTLR解释该动作来定义两个参数x和y：

public final AContext a(Map<String,String> x, int y)

throws RecognitionException

{

AContext \_localctx = new AContext(\_ctx, 0, x, y);

enterRule(\_localctx, RULE\_a);

...

}

**开始规则和EOF**

开始规则是解析器首先使用的规则; 它是语言应用程序调用的规则函数。

规则会选择匹配更多输入的匹配方案，请考虑以下匹配一个，两个或三个标记的规则，具体取决于输入。

s : ID

| ID '+'

| ID '+' INT

;

在a+3，规则s匹配第三个选择。

在a+b，它匹配第二个替代方案并忽略最终的b标记。

在a b，它匹配第一个替代，忽略b令牌。