Antlr4系列（一）：语法分析器学习

先锋
字节跳动 大数据后台开发
关注他

背景：因为系统的指标计算规则比较复杂，要用到规则引擎，然而发现没有一款既可以支持SQL表达式，又可以支持混合运算、逻辑运算等运算的开源规则引擎，于是决定自己实现一个。调研了一下，发现可以利用Antlr实现，于是花了一个周多一点时间学习了下，本文是学习记录，后续说明如何用Antlr实现规则引擎。

系列文章列表：

> **Antlr4系列（一）：语法分析器学习**
> Antlr4系列（二）：实现一个计算器
> Antlr4系列（三）：实现SQL Parser
> Antlr4系列（四）：实现一个表达式规则引擎

前言

语言由一系列有意义的语句组成，语句由词组和规则组成，词组又由词符号组成。对于一个输入，比如“sp=100"，我们需要将其识别为一个赋值语句，意味着需要知道sp是被赋值的目标，100是要被赋予的值，成功识别后，方能进行适当的操作。

在识别过程中，我们首先需要将字符聚集为单词或符号（词法符号，token），这个过程称为词法分析，关于其详细解释，维基百科解释如下：解释如下：

> 词法分析（英语：lexical analysis）是计算机科学中将字符序列转换为标记（token）序列的过程。进行词法分析的程序或者函数叫作词法分析器（lexical analyzer，简称lexer），也叫扫描器（scanner）。词法分析器一般以函数的形式存在，供语法分析器调用。
>
> 这里的标记是一个字串，是构成源代码的最小单位。从输入字符流中生成标记的过程叫作标记化（tokenization），在这个过程中，词法分析器还会对标记进行分类.

词法分析完成后，需要将这些单词或符号，根据语法规则，识别为有意义的语句，这个过程称为语法分析关于其详细解释，维基百科解释如下：

> 语法分析（英语：syntactic analysis，也叫 parsing）是根据某种给定的形式文法对由单词序列（如英语单词序列）构成的输入文本进行分析并确定其语法结构的一种过程。
>
> 语法分析器（parser）它的作用是进行语法检查、并构建由输入的单词组成的数据结构（一般是语法分析树、抽象语法树等层次化的数据结构）。语法分析器通常使用一个独立的词法分析器从输入字符流中分离出一个个的“单词”，并将单词流作为其输入。

更多解释：词法分析、语法分析、语义分析

简介

那么，Antlr是干嘛的了？

ANTLR（全称：ANother Tool for Language Recognition）是目前非常流行的语言识别工具，使用Java语言编写，基于LL(*)解析方式，使用自上而下的递归下降分析方法。通过输入语法描述文件来自动构造自定义语言的词法分析器、语法分析器和树状分析器等各个模块。ANTLR使用上下无关文法描述语言，文法定义使用类似EBNF的方式。

所有编程语言的语法，都可以用ANTLR来定义。ANTLR提供了大量的官方grammar示例，包含了各种常见语言，比如Java、SQL、Javascript、PHP等等。ANTLR的应用非常广泛，比如Hive、Presto和SparkSQL等的SQL Parser模块都是基于ANTLR构建的。

语言识别工具还有很多种，比如Lex和Yacc, 还有Apache Calcite里面使用的JavaCC等等。

JavaCC

JavaCC，即Java Cmopiler Compiler，为了简化基于Java语言的词法分析器或者语法分析器的开发，Sun公司的开发人员开发了JavaCC(Java Compiler Compiler)。

JavaCC是一个基于Java语言的分析器的自动生成器。用户只要按照JavaCC的语法规范编写JavaCC的源文件，然后使用JavaCC的编译器编译，就能够生成基于Java语言的某种特定语言的分析器。JavaCC已经成为最受欢迎的Java解析器创建工具。

YACC

YACC(Yet Another Compiler-Compiler): 1975 年由贝尔实验室 Mike Lesk & Eric Schmidt 开发，UNIX 标准实用工具 (utility)。

Antlr语法

语法模式

语言纷繁复杂，但是对其进行抽象，可以分为以下四种模式：

- 序列模式，即一系列元素
- 选择模式，在多个可选方案中做出选择
- 词法符号依赖，一个词法符号需要和某处的另外一个词法符号配对
- 嵌套结构，一种自相似的语言结构。

序列模式

它是一个任意长的，可能为空的序列，其中的元素可以是词法符号或者子规则。序列模式的例子包括变量声明和整数序列等等，比如：

```
'[' INT+ ']'  //Matlab的整数向量
```

带终止符的序列模式

它是一个任意长的，可能为空的序列，该序列由一个词法符号分隔开，通常是分号或换行符，其中的元素可以是词法符号或者子规则。这样的例子包括类C语言的语句集合和一些用换行符来分隔的数据格式。比如：

```
（statement ';')*  //java的语句集合
（row '\n')* //多行数据
```

带分隔符的序列模式

它是一个任意长的，可能为空的序列，该序列由一个词法符号分隔开，通常是逗号，分号或句号，其中
参数

函数调用的参数列表中找到这样的例子：

```
exprList: expr(',' expr)*
```

选择模式

它是一组备选分支的集合。使用 | 来分隔同一个语言规则的若干备选分支。这样的例子包括不同种类的类型、语句、表达式或者XML标签。比如下面这个字段定义表达式：

```
field：INT | STRING；   //允许子段中出现整数或者字符串
```

词法符号依赖模式

一个词法符号需要和一个或多个后续词法符号匹配。这样的例子包括配对的圆括号，花括号，方括号和尖括号。

比如对于整数向量序列[1,2,3]这种，为了描述向量左右两侧的方括号，需要一种方法来表达对这样的词法符号的依赖。此时，如果我们在一个语句中看到一个符号，那么就必须在语句中找到另一个配对的符号。为表达出这种语意，在语法中，使用一个序列来指明所有配对的符号，通常这些符号会把其他元素分组或包裹起来。

比如，对于一个整数向量，可以用这种方式表示：

```
vector: '[' INT+ ']' ;  //[1], [1,2], [1,2,3],...
```

以方法定义为例，它的语法表示如下：

```
functionDecl
    : type ID '{' formalParameters? '}'
    ;
formalParameters
    : formalParameter (',' formalParameter)*
    ;
formalParameter
    : type ID
```

嵌套模式

它是一种子相似的语言结构。这样的例子包括表达式，Java的内部类、嵌套的代码块以及嵌套的Python函数定义。

比如嵌套数组表达式定义：

```
expr: ID '[' expr ']'  //a[1],a[b[1]],a[(2*b[1])]
    | '(' expr ')'  //(1),(a[1]),((((1)))),(2*a[1])
    | INT
    ;
```

再比如类定义：

```
classDef:
    : 'calss' ID '{' (calssDef|method|field) '}' ;
```

具体语法

详细案例请看：Antlr Grammar，主要有下面的一些文法结构：

- 注释：和Java的注释完全一致，也可参考C的注释，只是增加了JavaDoc类型的注释；

- 标志符：参考Java或者C的标志符命名规范，针对Lexer 部分的 Token 名的定义，采用全大写字母的形式，对于parser rule命名，推荐首字母小写的驼峰命名；
- 不区分字符和字符串，都是用单引号引起来的，同时，虽然Antlr g4支持 Unicode编码（即支持中文编码），但是建议大家尽量还有英文；
- Action，行为，主要有@header 和@members，用来定义一些需要生成到目标代码中的行为，例如，可以通过@header设置生成的代码的package信息，@members可以定义额外的一些变量到Antlr4语法文件中；
- Antlr4语法中，支持的关键字有：import, fragment, lexer, parser, grammar, returns, locals, throws, catch, finally, mode, options, tokens。

整体结构如下：

```
/** Optional javadoc style comment */

grammar Name;

options {...}

import ... ;
tokens {...}

channels {...} // lexer only

@actionName {...}
rule1 // parser and lexer rules, possibly intermingled

...

ruleN
```

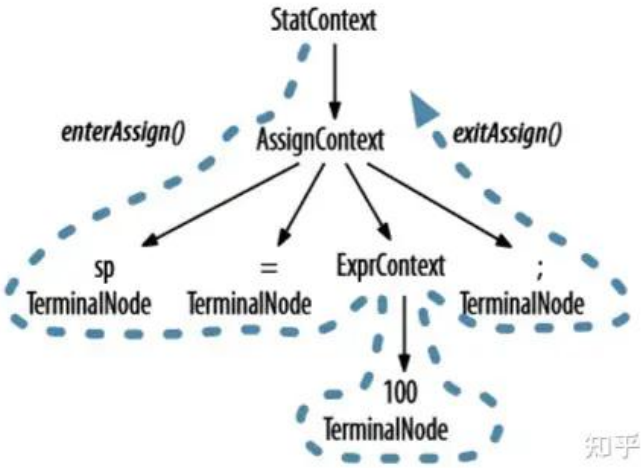
一般如果语法非常复杂，会基于Lexer和Parser写到两个不同的文件中（例如Java，可参考：https://github.com/antlr/grammars-v4/tree/master/java/java8），如果语法比较简单，可以只写到一个文件中（例如Lua，可参考：https://github.com/antlr/grammars-v4/blob/master/lua/Lua.g4）。

监听器和访问器

ANTLR除能够自动构建语法分析树外，还能生成基于Listener（监听者模式，通过节点监听，触发处理方法）和Visitor（访问者模式，主动遍历）的树遍历器。访问者模式遍历语法树是一种更加灵活的方法，可以避免在文法文件中嵌入繁琐的动作，使解析与应用逻辑代码分离，这样不但文法的定义更加简洁清晰，而且可以在不重新编译生成语法分析器的情况下复用相同的语法，甚至能够采用不同的程序语言来实现这些动作。

Listener监听器

ANTLR为每个语法文件生成一个ParseTreeListener的子类，在该类中，语法中的每条规则都有对应的enter方法和exit方法，当访问到某节点时，就会调用相应的enter方法，访问完后就调用相应的exit方法。下图是ParseTreeWalker对一个简单语法分析树进行深度优先遍历的过程。

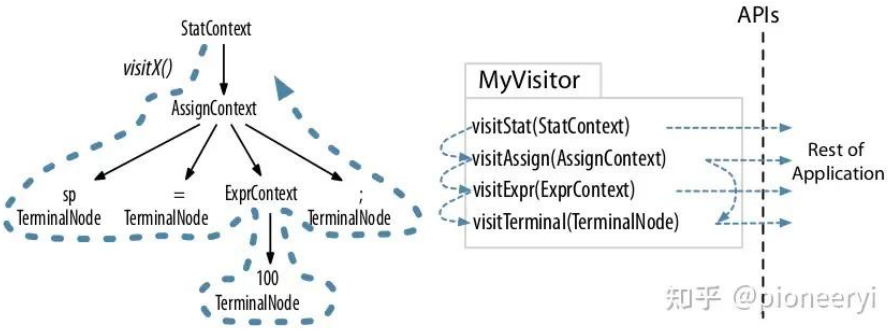


ParseTreeWalker对监听器方法的完整的调用顺序如下图所示：

Visitor访问器

访问者模式（Visitor Pattern）是一种将操作与对象结构分离的软件设计模式，提供作用于某种对象结构上各元素的操作，可以使我们在不改变元素结构的前提下，定义作用于元素的新操作。

这种模式的工作方法如下：假设有一个由许多元素Node构成的对象结构Tree，这些Node类都拥有一个accept方法用来接受访问者对象Visitor的访问；Visitor类是一个接口，它拥有一个visit方法，这个方法对访问到的Tree中不同类型的Node作出不同的反应；在对Tree的一次访问过程中会遍历整个Tree，对遍历到的每个Node都调用accept方法，在每个元素的accept方法中回调Visitor的visit方法，从而使Vistior得以处理Tree的每个Node；可以针对Tree设计不同的Visitor实现类来完成不同的操作。



Antlr入门案例

以一个最简单的例子：匹配关键字hello和标志符，来了解一下Antlr的应用。首先定义语法和词法，创建一个.g4文件，用于定义词法分析器（lexer）和语法解析器(Parser)，g4文件内容如下：

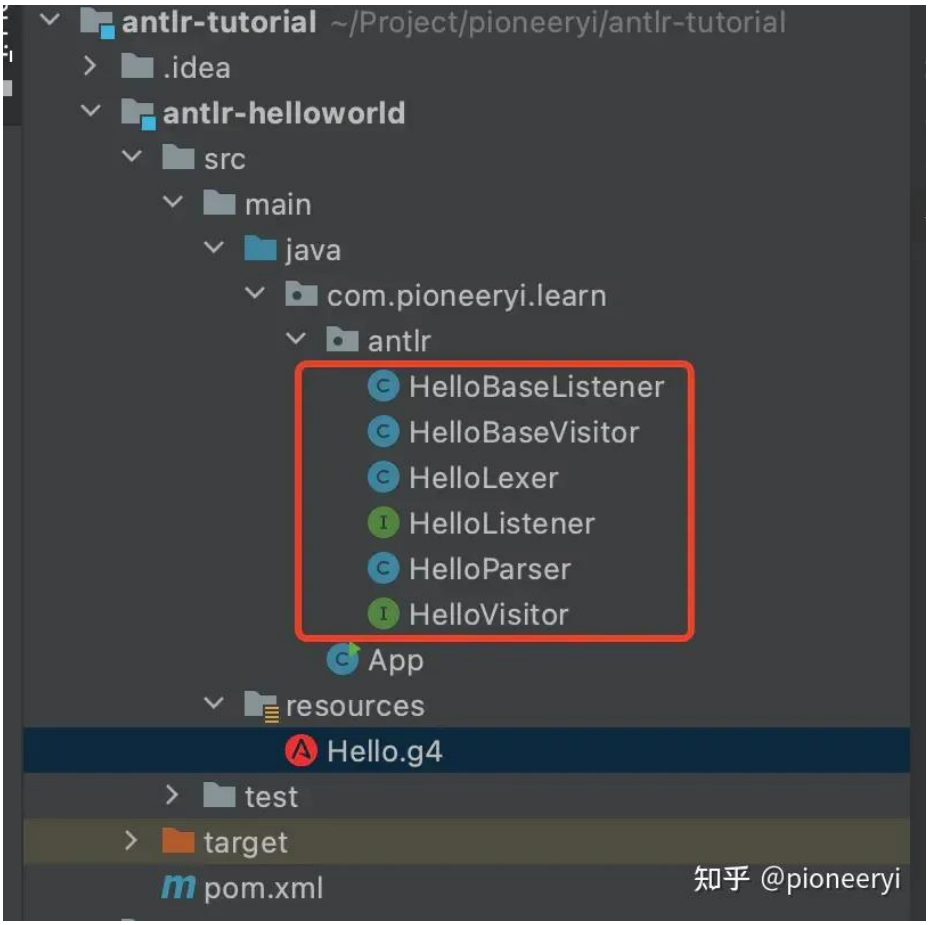
```
grammar Hello;              // 1、定义文法的名字
@header { package com.pioneeryi.learn.antlr; }  //2、java package

s
```

```
ID : [a-z]+ ;              // 标志符由小写字母组成
WS : [ \t\r\n]+ -> skip ;  // 4、跳过空格、制表符、回车符和换行符
```

其中，
1、定义了 grammar 的名字，名字需要与文件名对应
2、定义生成的Java类的package
3、s定义的语法，会使用到下方定义的正则表达式词法
4、定义了空白字符，后面的 skip 是一个特殊的标记，标记空白字符会被忽略。

在IDEA中右键点击.g4文件，选择Generate ANTLR Recognizer，插件会自动在gen目录下生成一堆Java代码，我们需要移动到对应的package中。如果定义了@header，IDEA也会自动生成package信息，可以看到上面g4会自动生成如下文件：



我们可以利用下面这段代码来测试一下ParseTree：

```
public class HelloTest {
    public static void main(String[] args) throws Exception {
        HelloLexer lexer = new HelloLexer(CharStreams.fromString("hello world"));
        CommonTokenStream tokens = new CommonTokenStream(lexer);
        HelloParser parser = new HelloParser(tokens);
        ParseTree tree = parser.s();
        System.out.println(tree.toStringTree(parser));
    }
}
```

参考文档

1、Antlr官方文档

2、《Antlr 4权威指南》

编辑于 2023-10-18 20:43 · IP 属地广东