Antrl简介

Antrl—Another Tool for Language Recognition，包括Java,C++,C#在内的语言提供了一个通过语法描述来自动构造自定义的识别器、编译器（parser）和解释器（translator）的框架。Antrl通过断言（Predicate）解决识别冲突；支持动作（Action）和返回值（Return Value）来；可以根据输入自动生成语法树并可视化的显示出来。

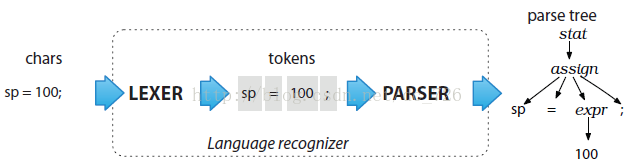
# 简介

实现一种语言时，需要构建读取句子的应用，并对输入中的元素做出反应。如果应用计算或执行句子，就叫解释器，包括计算器、配置文件读取器、Python解释器都属于解释器。翻译器是指将句子转换成另一种语言，如JAVA到C#的翻译器和编译器都属于翻译器。不管是解释器还是翻译器，应用首先要识别出所有有效的句子、词组及字词组等，识别语言的程序就是解析器（parser）或语法分析器（syntax analyzer）。

ANTLR提供一种编写解析器的方法，只需要编写ANTLR的语法文件，描述我们要解析的语言的语法，之后ANTLR就会自动生成解析各种语言的解析器，也就是说ANTRL是一种能写出程序的程序。ANTRL解析语言分析两个阶段：

1. 词法分析（lexical analysis），对应的分析程序叫做lexer，负责将符号（token）分组成符号类（token class 或者token type）
2. 第二个阶段是parser，即语法分析，构建一棵分析树（parse tree）或者语法树（syntax tree）

如下图例子，将简单的赋值表达式进行解析的过程：

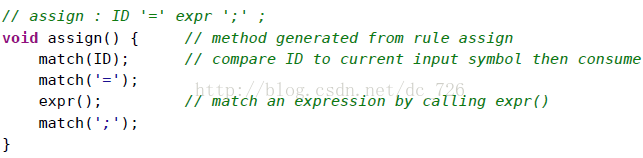


语法树的叶子是输入token，而上级结点是包含其孩子结点的词组名（phase），线性的句子其实是语法树的序列化。最终生成语法树的好处是：

1. 树形结构易于遍历和处理，并且易被程序员理解，方便应用代码做进一步处理
2. 多种解析或者翻译的应用代码都可以重用一个解析器。ANTRL也支持像传统解析器生成器那样，将应用处理代码嵌入到语法中
3. 对于计算依赖而需要多趟处理的翻译器来说，语法树很有用。不需要多次调用解析器去解析，只需高效的遍历语法树多次。

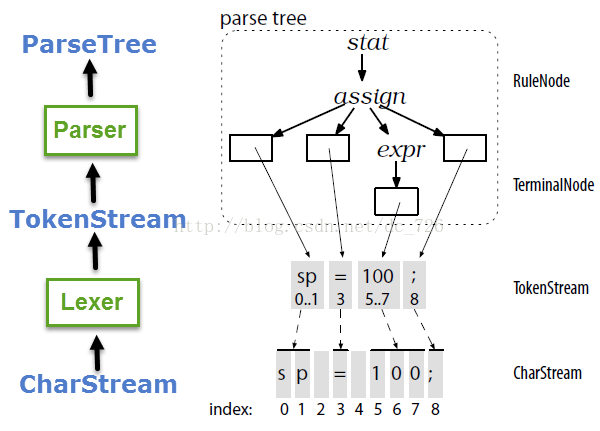
# 深入Antlr

ANTLR生成的解析器叫做递归下降解析器（recursive-descent parser），属于自顶向下解析器（top-down parser）的一种，解析过程中从语法树的根开始向叶子（token）递归，这点与代码的调用图与树节点可以对应上。以上面的赋值表达式解析为例，其递归下降解析器的代码大概如下：

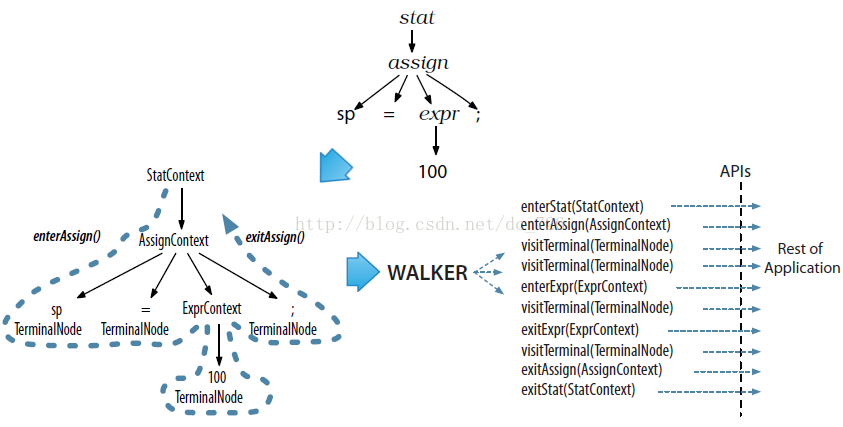


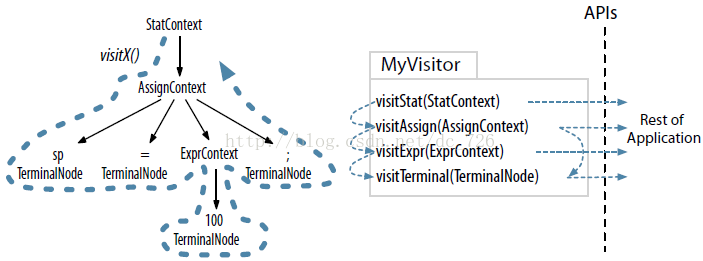
Assign很简单，直接顺序读取输入字符，不做任何选择。

在内部，ANTLR的数据结构会共享数据来节约内存，如下图所示，解析树的叶子节点指向Token流中的Token，而Token中的起止字符索引指向字符流，而非拷贝子字符串。而像空格这种不与任何Token相关的字符会直接被Lexer丢弃掉：



ANTLR为每个Rule都生成一个Context对象，它会记录识别时的所有信息。ANTLR提供了Listener和Visitor两种遍历机制。Listener是全自动化的，ANTLR会主导深度优先遍历过程，只需要处理各种事件就可以了。而Visitor则提供了可控的遍历方式，可以自行决定是否显示调用子结点的visit方法：





# Antlr 示例

下面是一个例子，将short[] data = {1, 2, 3} 翻译成 string data = "\u0001\u0002\u0003"

1. 实现解析器，编写.g4文件，创建一个文件，后缀名为g4

*grammar ArrayInit;*

*init : '{' value (',' value)\* '}' ;*

*value : init| INT;*

*INT : [0-9]+ ;*

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

1. 使用maven plugin生成代码

在项目pom中添加maven依赖：

*<dependency>*

*<groupId>org.antlr</groupId>*

*<artifactId>antlr4-runtime</artifactId>*

*<version>4.7</version>*

*</dependency>*

Antlr plugin：

*<build>*

*<plugins>*

*<plugin>*

*<groupId>org.antlr</groupId>*

*<artifactId>antlr4-maven-plugin</artifactId>*

*<version>4.7</version>*

*<executions>*

*<execution>*

*<id>antlr</id>*

*<goals>*

*<goal>antlr4</goal>*

*</goals>*

*</execution>*

*</executions>*

*<configuration>*

*<outputDirectory>${basedir}/src/main/java</outputDirectory>*

*<sourceDirectory>${basedir}/src/main/java</sourceDirectory>*

*<listener>true</listener>*

*<visitor>true</visitor>*

*<treatWarningsAsErrors>true</treatWarningsAsErrors>*

*</configuration>*

*</plugin>*

*</plugins>*

*</build>*

编译命令如下：

*mvn clean package*

生成代码结构，目录如下：

*|antlrtest*

*|—com.fys*

*|—ArrayInit.g4*

*|—ArrayInitBaseListener*

*|—ArrayInitBaseVisitor*

*|—ArrayInitLexer*

*|—ArrayInitListener*

*|—ArrayInitParser*

*|—ArrayInitVisitor*

*|—ArrayInit.tokens*

1. 编写应用程序

*public class ShortToUnicodeString extends ArrayInitBaseListener {*

*@Override*

*public void enterInit(ArrayInitParser.InitContext ctx) {*

*System.out.print('"');*

*}*

*@Override*

*public void exitInit(ArrayInitParser.InitContext ctx) {*

*System.out.print('"');*

*}*

*@Override*

*public void enterValue(ArrayInitParser.ValueContext ctx) {*

*if (ctx.INT() == null) {*

*System.out.print(ctx.INT());*

*} else {*

*System.out.printf("\\u%04x", Integer.valueOf(ctx.INT().getText()));*

*}*

*}*

*}*

*public class Main {*

*public static void main(String[] args) {*

*String sentence = "{99, 3, 451}";*

*ArrayInitLexer lexer = new ArrayInitLexer(new ANTLRInputStream(sentence));*

*CommonTokenStream tokens = new CommonTokenStream(lexer);*

*ArrayInitParser parser = new ArrayInitParser(tokens);*

*ParseTree tree = parser.init();*

*ParseTreeWalker walker = new ParseTreeWalker();*

*walker.walk(new ShortToUnicodeString(), tree);*

*}*

*}*

执行Main程序，结果如下：

*"\u0001\u0002\u0003"*