Ch02 The Big Picture

Overview

要实现一种语言，我们需要创建一个程序，它能够读取“句子”，对其中的“短语”和“词”做出合适的“反应”。大体来说，如果一个程序直接“计算”或“执行”句子，我们称之为“**解释器（interpreter）**”，如计算器、Python解释器等。如果一个程序将句子从一种语言转换为另一种，我们称之为“**翻译器（translator）**”，如Java2C#转换器或**编译器**。

想象理解一个自然语言中的句子，如“他喜欢编程语言”。我们识别出这是一个陈述句（而不是疑问句或祈使句等），其中的每个词及其词性、句子成分（主谓宾补定状）。

理解编程语言大致类似。如Java中的语句

```
sp = 1;
```

我们说这是一条赋值语句（而不是import或class等），进一步地，sp是赋值目标，1是欲赋的值。程序识别到这里，可执行一个合适的操作，如：performAssignment("sp", 1)。

识别语言的程序成为**解析器**（parser，或语法分析器syntax analyzer）。对于一个ANTLR应用，它需要创建语法（grammar）。语法即是一组规则，每条规则都表示了某种特定的结构。ANTLR将语法翻译为解析器。当然ANTLR grammar语言本身也要遵循一定的语法，故称为ANTLR's meta-language。

解析过程可分解为**两个类似有所区别的任务或阶段**，这种分解会大大简化解析器的工作。这很像我们在读中文文本时，大脑会潜意识地先分词再理解词之间的关系与语义。

分词阶段对应于词法分析（lexical analysis，或tokenizing），进行词法分析的工具称为**lexer**。lexer将每个token归为相应的类别（token type），如INT、IDENTIFIER、FLOAT等。也就是lexer的结果是，我们知道了每个**token的类型与内容（text）**。

第二个阶段自然就是具体的解析了。ANTLR生成的parser会为输入的句子创建出称为**解析树（parse tree）**或**语法树（syntax tree）**的数据结构。解析树包含了token/symbol的完整信息，**后续阶段易于处理**。

Implementing Parsers

ANTLR根据语法定义生成递归下降（recursive-descent）解析器，这种解析器属于自顶向下解析法的一种。

```
void assign():
    match(ID);
    match('=');
    expr();
    match(';')
```

match方法对应到解析树的叶节点。上面assign方法可确保所需要的token都以正确的顺序出现。考虑到一种语法中会包含多种语句（alternatives），故stat规则会先选择出匹配的子规则，其选择基于发现的token序列类型，如：

```
void stat() {
    switch (<<current token>>) {
        case ID: assign(); break;
        case IF: ifstat(); break;
        ...
        default: <<raise exception>>
```

一旦确定了接下来的规则是assign，进入assign方法，此时就不需要考虑其它规则的情况了。

stat方法需要作出一个**parsing decision**或prediction，上述伪代码显示的是通过当前看到的token，但如果仅由一个token不足以确定，那么就需要预先查看更多的token，即**lookahead token**。

这个预先探测的过程，如同探测一个迷宫的出口。我们需要在所有可能的路径作出选择。如果当前位置只有一条路可走，那么可以当即作出选择，否则就需要再往前走走看，即lookahead，直到可以确定出一条可行的路。值得注意的是，有效的路径可能不止一条。

歧路的选择

如果有多条路径可选，就说这一句子有了**歧义（ambiguity）**。如：

`for whom no thanks is too much.`

自然语言中出现的歧义或许是一种趣味所在，但对于编程语言的解析器来说，它需要确定出唯一的一个规则。

大多数语言的设计者都希望他们的语言是无歧义的，故包含歧义的语法可认为是一个bug。ANTLR解决歧义的方法是，选择第一条有效规则。歧义在lexer和parser中都可能出现，如多数语言中都有keyword和identifier之分：

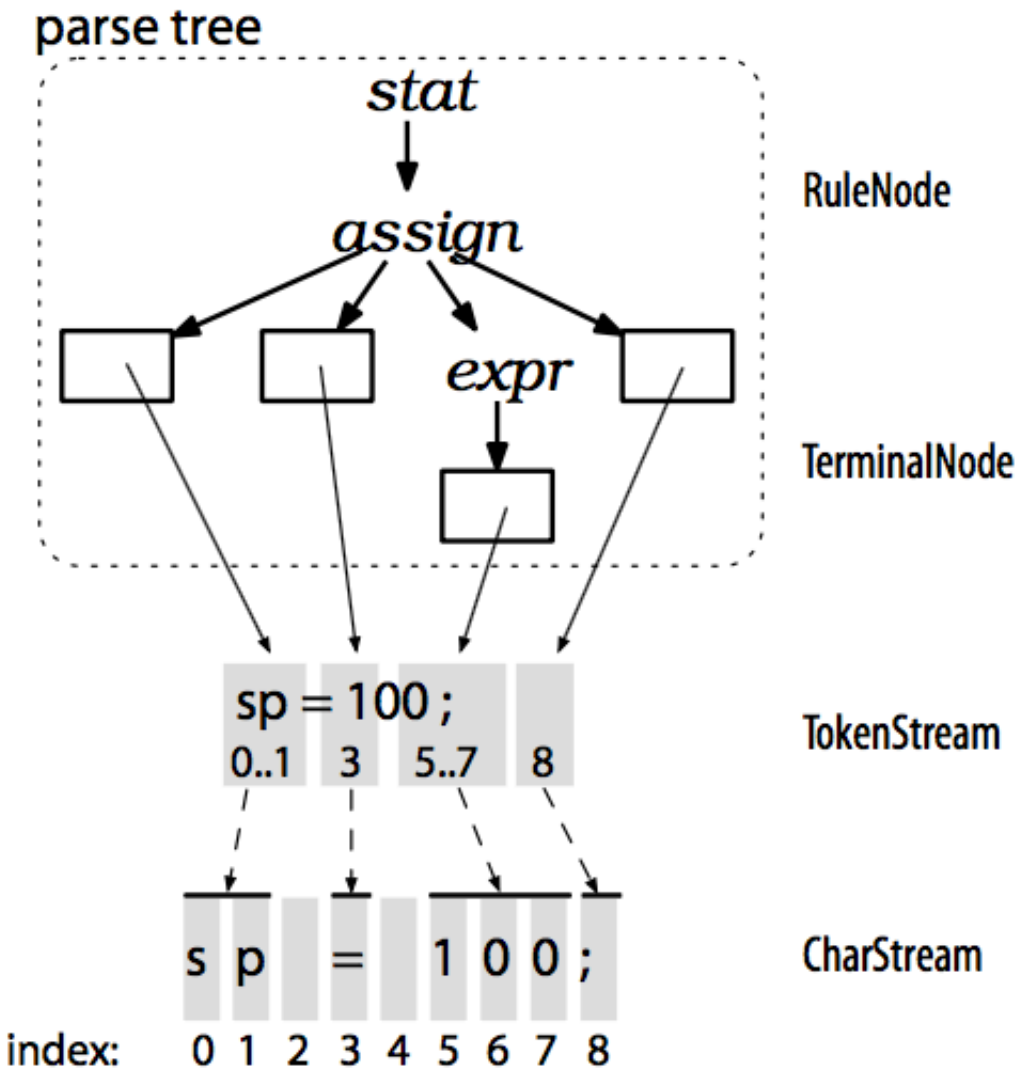
```
BEGIN : 'begin';
ID    : [a-z]+;
```

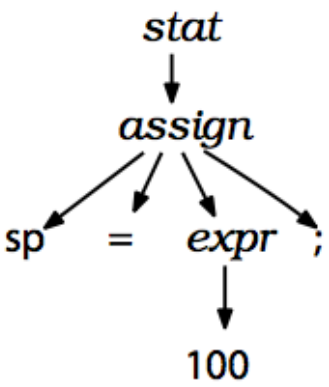
ANTLR选择其中出现的第一条，即keyword，这样在定义语法时，顺序变得重要。另外，lexer会尝试匹配最大长度的字符串，所以beginner就只能是ID了。

在语言应用中使用解析树

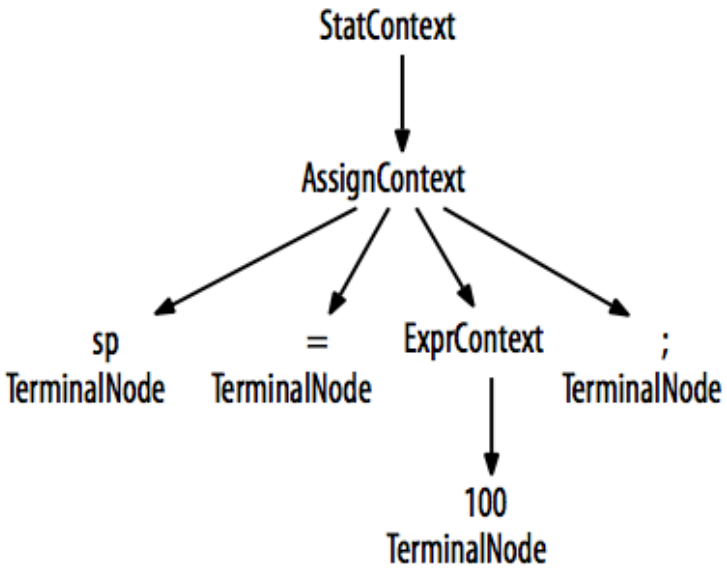
一个语言应用，需要对指定的语言输入执行所需的功能。由于ANTLR能够生成parser，所以我们只需要了解如何操作解析树。

在ANTLR中，从输入到解析树涉及的类有CharStream、Lexer、Token、Parser和ParseTree。连接lexer和parser的pipe称为TokenStream。


parse tree
stat
assign
expr
RuleNode
TerminalNode
sp = 100 ;
0..1 3 5..7 8
TokenStream
s p = 1 0 0 ;
CharStream
index: 0 1 2 3 4 5 6 7 8



Parse tree

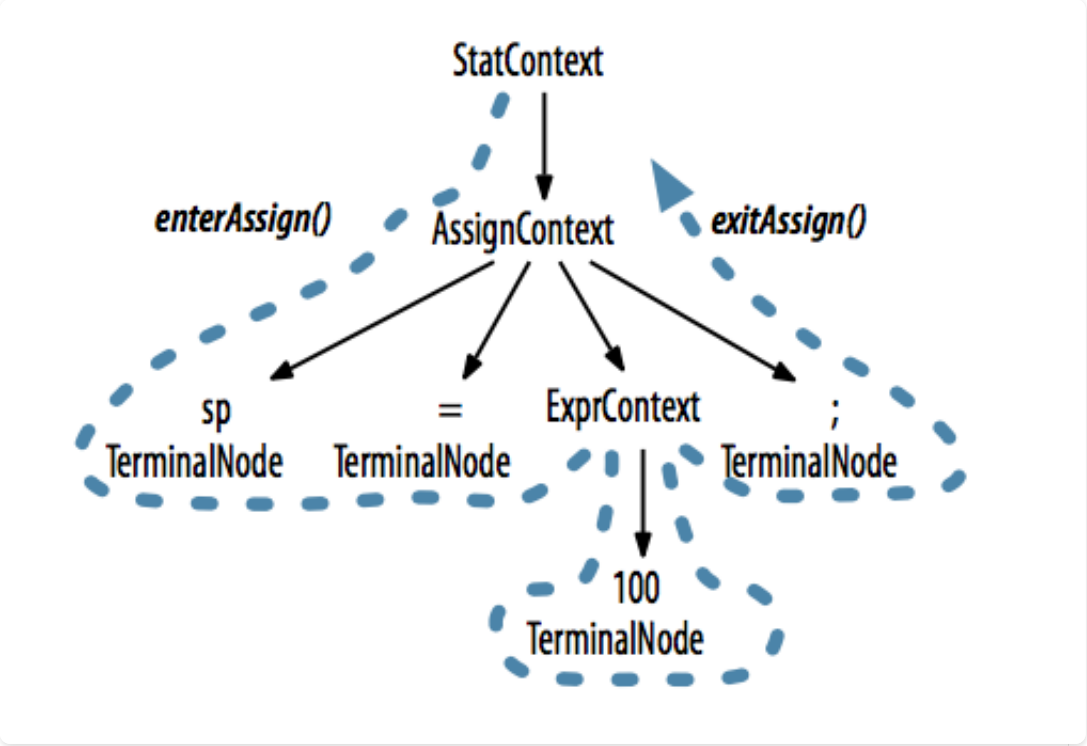


Parse tree node class names

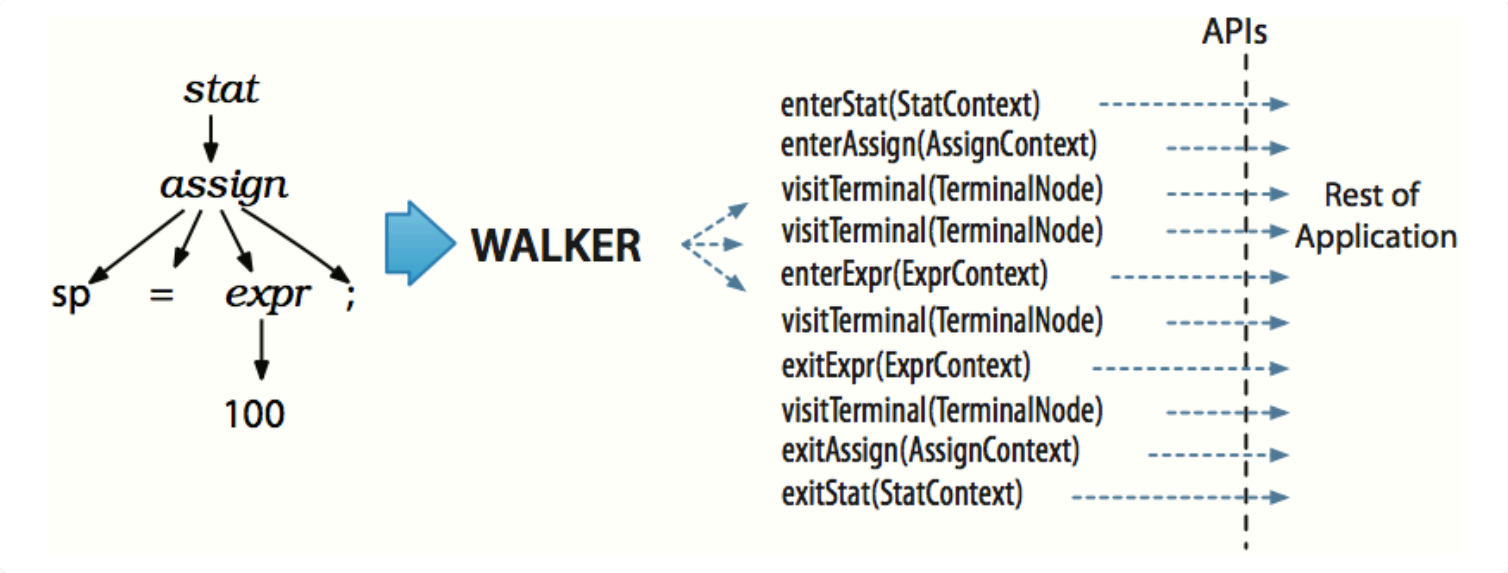
这些context对象保存了每条rule的上下文信息。有了这些信息，我们可以对解析树进行深度优先的遍历。

解析树的Listener和Visitor

ANTLR提供了两种遍历机制。一是listener接口，可对内置的树遍历事件进行响应，类似SAX doc之于XML parser。



当遍历者（walker）遇见assign规则的node，会触发`enterAssign()`事件，将其传送给`AssignContext`，当遍历者访问了`assign`的所有子节点后，触发`exitAssign()`。从上图可以看到，遍历过程是深度优先的。



listener机制的优势之处在于，它是完全自动的，我们不需要编写任何遍历代码。

Parse-Tree Visitors

如果希望控制遍历的过程，可以使用visitor模式。ANTLR的`-visitor`选项可生成visitor接口代码。