Atitit 表达式原理 语法分析 原理与实践 解析java的dsl 递归下降是现阶段主流的语法分析方法

[于是我们可以把上面的语法改写成如下形式： 1](#_Toc32351)

[合并前缀 1](#_Toc9788)

[语法分析有自上而下和自下而上两种分析方法 2](#_Toc12906)

[递归下降是现阶段主流的语法分析方法， 2](#_Toc17474)

## 于是我们可以把上面的语法改写成如下形式：

1)       Operator=”+” | “-” | “\*” | “/”

2)       Expressio=<数字> | “(“ Expressio “)” | “(“ Operator Expressio Expressio “)”

1. 语法规则说的是Operator，也就是操作符，可以是加号、减号、乘号或者除号。第二条语法规则说的是一条表达式可以只由数字构成、一个加了括号的表达式或者一个加上了括号的操作符和两个参数。

## ****合并前缀****

“(“ Expressio “)” | “(“ Operator Expressio Expressio “)”。我们注意到，这两个部分都是使用括号开始和结束的，因此在写代码的时候可以把它们写在一起，只把中间的部分分开。这种方法在课本中通常被称为**合并前缀**。于是我们可以写一个GetExpressio函数。这个函数首先判断字符串是不是由数字开头，否则的话看一看是否由括号开头。如果是括号开头的话，那么检查接下来的是Operator还是一个Expressio。如果是Expressio则到此结束，如果是Operator的话还要再输入两个Expressio。然后判断一下是不是由右括号结束字符串：

篇文章相比起以前的两篇正则表达式来的确是短了不少。递归下降法是一种适合人脑使用而不是电脑使用的方法。这种方法非常好用，所以大部分编译原理的教科书都会专门使用一个章节来说明递归下降的实现、局限性以及遇到的问题的解决方法。这篇文章不是理论文章，所以有一些本文没阐述到的问题可以通过人的智商来解决。

。今天介绍的这种方法叫做**递归下降**（recursive descent）法，这是一种适合手写语法编译器的方法，且非常简单。递归下降法对语言所用的文法有一些限制，但递归下降是现阶段主流的语法分析方法，因为它可以由开发人员高度控制，在提供错误信息方面也很有优势。就连微软C#官方的编译器也是手写而成的递归下降语法分析器。

## 语法分析有自上而下和自下而上两种分析方法

其中  
自上而下：递归下降，LL(1)  
自下而上：LR(0),SLR(1),LR(1),LALR(1)

## 递归下降是现阶段主流的语法分析方法，

因为它可以由开发人员高度控制，在提供错误信息方面也很有优势。

。今天介绍的这种方法叫做**递归下降**（recursive descent）法，这是一种适合手写语法编译器的方法，且非常简单。递归下降法对语言所用的文法有一些限制，但递归下降是现阶段主流的语法分析方法，因为它可以由开发人员高度控制，在提供错误信息方面也很有优势。就连微软C#官方的编译器也是手写而成的递归下降语法分析器。

化成代码就是这样：

|  |
| --- |
| Node ParseNode()  {  int lookAheadIndex = m\_index;  char lookAheadChar = m\_inputString[lookAheadIndex];  if (Char.IsLetter(lookAheadChar))  {  //采用N → a(N, N)继续分析  }  else if (lookAheadChar == ',' || lookAheadChar == ')' )  {  //采用N → ε继续分析  }  else  {  throw new Exception("语法错误");  } |

下面我们要研究一下递归下降法对文法有什么限制。首先，我们必须要通过超前查看进行分支预测。**支持递归下降的文法，必须能通过从左往右超前查看k个字符决定采用哪一个产生式**。我们把这样的文法称作**LL(k)**文法

/AtiPlatf\_cms/src/com/attilax/ast/AstBuilder.java

List<Token> tokens;

**int** m\_index=0;

**public** Expressio buildAstV2(List<Token> tokens) {

**this**.tokens = tokens;

// System.out.println(AtiJson.toJson(tokens));

List<String> tokens\_slice\_li = Lists.*newLinkedList*();

String stat = "ini";

MethodInvocation mi = **new** MethodInvocation();

Expressio exp;

**boolean** isConstrutsced = **false**; // qaa is first

**boolean** isNewDysnInvoke = **false**;

**boolean** isStaticInvoke = **false**;

Token cur = tokens.get(m\_index);

**if** (cur.getText().equals("(") && cur.getType().equals("op"))

{

stat = "brkStart";

String text = tokens.get(m\_index - 1).getText();

//newxx first

**if** (isNewOpV2(tokens) && isConstrutsced == **false**)// newxxx

{

String className = javaUtil.*clr*(text);

mi.Exp = getClassInstance(className);

**while** (**true**) {

m\_index++;

cur = tokens.get(m\_index);

**if** (cur.getText().equals("(") && cur.Type.equals("op")) {

mi.Name=tokens.get(m\_index-1).getText();

//return mi;

**break**;

}

}

// param push

**while** (**true**) {

m\_index++;

cur = tokens.get(m\_index);

**if** (cur.getText().equals(")") && cur.Type.equals("op")) {

**break**;

}

mi.arguments.add(cur.Text);

}

**return** mi;

}

**if** (isConstrutsced == **false**) { // static //first

// new method invoke exp

String className = refx.*getClassName*(text);

className = javaUtil.*clr*(className);

mi.Exp = **new** SimpleName(className); // cls name

mi.Name = refx.*getMethodName*(text);

isConstrutsced = **true**;

// param push

**while** (**true**) {

m\_index++;

cur = tokens.get(m\_index);

**if** (cur.getText().equals(")") && cur.Type.equals("op")) {

**break**;

}

mi.arguments.add(cur.Text);

}

**return** mi;

}

// new xxx() meth.chain

/\*

\* if(mi.Name.equals("\_\_con")) mi.Name=text; else //static ,meth

\* chain { MethodInvocation tmp=(MethodInvocation) mi.clone();

\* MethodInvocation mi\_Outer\_new = new MethodInvocation();

\* mi\_Outer\_new.Exp=tmp; mi\_Outer\_new.Name=text; mi=mi\_Outer\_new; }

\* tokens\_slice\_li = Lists.newLinkedList(); continue;

\*/

// tokens\_slice\_li = Lists.newLinkedList();

// continue;

}

m\_index++;

**return** buildAstV2(tokens);

}

自己动手开发编译器（七）递归下降的语法分析器 - 装配脑袋 - 博客园.html

作者:: 绰号:老哇的爪子 （ 全名：：Attilax Akbar Al Rapanui 阿提拉克斯 阿克巴 阿尔 拉帕努伊 ）

汉字名：艾提拉（艾龙），   EMAIL:1466519819@qq.com

转载请注明来源： http://blog.csdn.net/attilax

Atiend