

让我们构建一个简单的解释器。第 8 部分。 (<https://ruslanspivak.com/lsbasi-part8/>)

日期 2016 年 1 月 18 日, 星期一

今天我们将讨论**一元运算符**，即一元加 (+) 和一元减 (-) 运算符。

今天的很多资料都基于上一篇文章中的资料，因此如果您需要复习，请返回第 7 部分 (<http://ruslanspivak.com/lsbasi-part7/>)并再次阅读。请记住：重复是所有学习之母。

话虽如此，这就是你今天要做的：

- 扩展语法以处理一元加号和一元减号运算符
- 添加一个新的UnaryOp AST节点类
- 扩展解析器以生成具有UnaryOp 节点的AST
- 扩展解释器并添加一个新的visit_UnaryOp方法来解释一元运算符

让我们开始吧，好吗？

到目前为止，我们只使用了二元运算符 (+、-、*、/)，即对两个操作数进行运算的运算符。

那什么是一元运算符呢？一元运算符是 仅对一个操作数进行运算的运算符。

以下是一元加号和一元减号运算符的规则：

- 一元减号 (-) 运算符产生其数值操作数的否定
- 一元加号 (+) 运算符生成其数字操作数而无需更改
- 一元运算符的优先级高于二元运算符 +、-、* 和 /

在表达式 “+ - 3” 中，第一个 '+' 运算符代表一元加运算，第二个 '-' 运算符代表一元减运算。表达式 “+ - 3” 等价于 “+ (- (3))”，它等于 -3。也可以说表达式中的 -3 是一个负整数，但在我们的例子中，我们将其视为一元减运算符，其中 3 作为其正整数操作数：

$$\begin{array}{c} + \quad - \quad 3 \\ \uparrow \quad \uparrow \\ \text{unary plus} \quad \text{unary minus (negation)} \end{array} = + (- (3)) = -3$$

我们再来看看另一个表达式，“5 - - 2”：

$$5 - - 2 = 5 - (- (2)) = 5 - (-2) = 7$$

binary minus (subtraction) unary minus (negation)

在表达式 “5 - - 2” 中，第一个 '-' 代表二元减法运算，第二个 '-' 代表一元减法运算，即否定。
 还有一些例子：

$$5 + - 2 = 5 + (- (2)) = 5 + (-2) = 3$$

binary plus (addition) unary minus (negation)

$$5 - - - 2 = 5 - (- (- (2))) = 5 - (- (-2)) = 5 - 2 = 3$$

binary minus (subtraction) unary minus (negation) unary minus (negation)

现在让我们更新我们的语法以包含一元加号和一元减号运算符。我们将修改因子规则并在那里添加一元运算符，因为一元运算符的优先级高于二元 +、-、* 和 / 运算符。

这是我们当前的因子 规则：

$$\text{factor} : \text{INTEGER} \mid \text{LPAREN expr RPAREN}$$

这是我们处理一元加和一元减运算符的更新因子规则：

$$\text{factor} : (\text{PLUS} \mid \text{MINUS}) \text{factor} \mid \text{INTEGER} \mid \text{LPAREN expr RPAREN}$$

如您所见，我将因子规则扩展为引用自身，这使我们能够推导出诸如 “- - - + - 3” 之类的表达式，这是一个带有许多一元运算符的合法表达式。

这是现在可以使用一元加号和一元减号运算符派生表达式的完整语法：

```

expr : term ((PLUS | MINUS) term) *
term : factor ((MUL | DIV) factor) *
factor : (PLUS | MINUS) factor | INTEGER | LPAREN expr RPAREN

```

下一步是添加一个AST节点类来表示一元运算符。

这个会做：

```

class UnaryOp ( AST ):
    def __init__ ( self , op , expr ):
        self . 令牌 = 自我。操作 = 操作
        自我。expr = expr

```

构造函数接受两个参数：op，它代表一元运算符标记（加号或减号）和expr，它代表一个AST 节点。

我们更新的语法对因子规则进行了更改，因此这就是我们要在解析器中修改的内容 - 因子方法。我们将添加代码来处理方法 “ (PLUS | MINUS) 因子” 分治：

```

def factor ( self ):
    """factor : (PLUS | MINUS) factor | INTEGER | LPAREN expr RPAREN"""
    token = self . current_token
    如果是 token . 类型 == PLUS :
        自我。吃 (PLUS )
        节点 = UnaryOp ( 令牌, 自。因子 () )
        返回 节点
    的 elif 令牌。类型 == 减号:
        自我。吃 (减)
        node = UnaryOp ( token , self . factor () )
        返回 node
    elif token . 类型 == 整数:
        自我。吃 (INTEGER )
        返回 Num (令牌)
    elif 令牌。类型 == LPAREN :
        self . 吃 (LPAREN )
        节点 = 自我。expr ()
        自我。吃 (RPAREN )
        返回 节点

```

现在我们需要扩展Interpreter类并添加一个visit_UnaryOp方法来解释一元节点：

```

def visit_UnaryOp ( self , node ):
    op = node . 操作。输入
    if op == PLUS :
        return + self . 访问 (节点。EXPR )
    elif 的 运算 == 减号:
        回报 -自我。访问 (节点。EXPR )

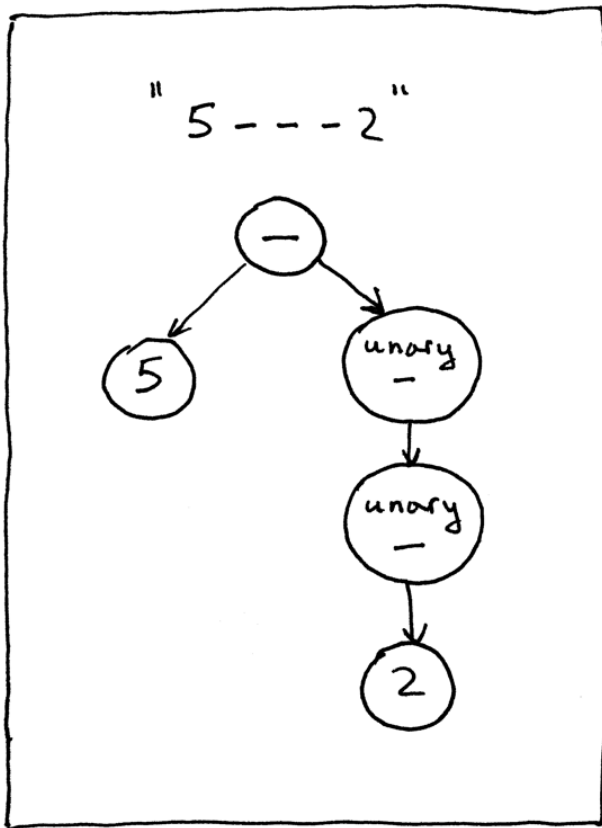
```

向前！

让我们手动为表达式 “5 - - 2” 构建一个AST，并将其传递给我们的解释器以验证新的visit_UnaryOp方法是有效。以下是从 Python shell 执行此操作的方法：

```
>>> from spi import BinOp, UnaryOp, Num, MINUS, INTEGER, Token
>>> Five_tok = Token ( INTEGER, 5 )
>>> two_tok = Token ( INTEGER, 2 )
>>> minus_tok = Token ( MINUS, '-' )
>>> expr_node = BinOp (
...     Num ( Five_tok ),
...     minus_tok,
...     UnaryOp ( minus_tok, UnaryOp ( minus_tok, Num ( two_tok ) ) )
... )
>>> from spi import Interpreter
>>> inter = Interpreter ( None )
>>> inter.访问( expr_node )
3
```

从视觉上看，上面的AST树是这样的：



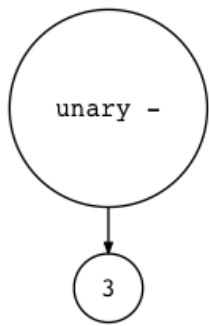
直接从GitHub (<https://github.com/rspivak/lbasi/blob/master/part8/python/spi.py>)下载本文解释器的完整源代码。亲自尝试一下，看看您更新的基于树的解释器是否正确评估包含一元运算符的算术表达式。

这是一个示例会话：

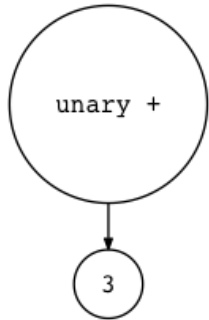
```
$ python spi.py
spi> - 3
-3
spi> + 3
3
spi> 5 - - - + - 3
8
spi> 5 - - - + - ( 3 + 4 ) - +2
10
```

我还更新了genastdot.py (<https://github.com/rspivak/lbasi/blob/master/part8/python/genastdot.py>)实用程序来处理一元运算符。以下是为带有一元运算符的表达式生成的AST图像的一些示例：

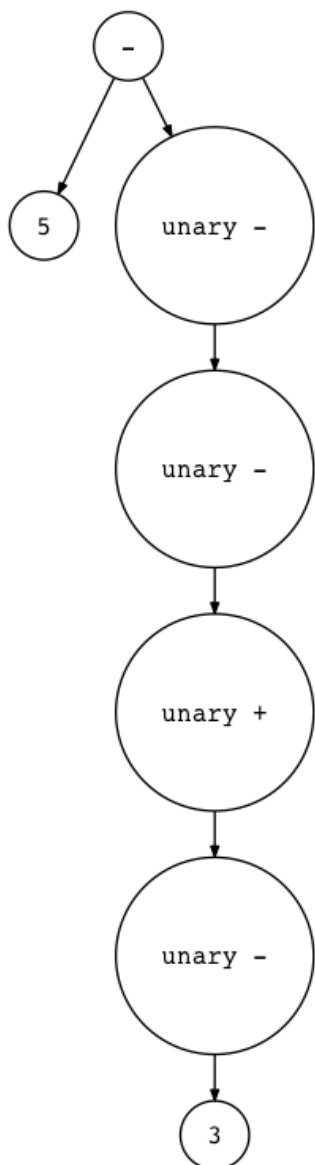
```
$ python genastdot.py "- 3" > ast.dot && dot -Tpng -o ast.png ast.dot
```



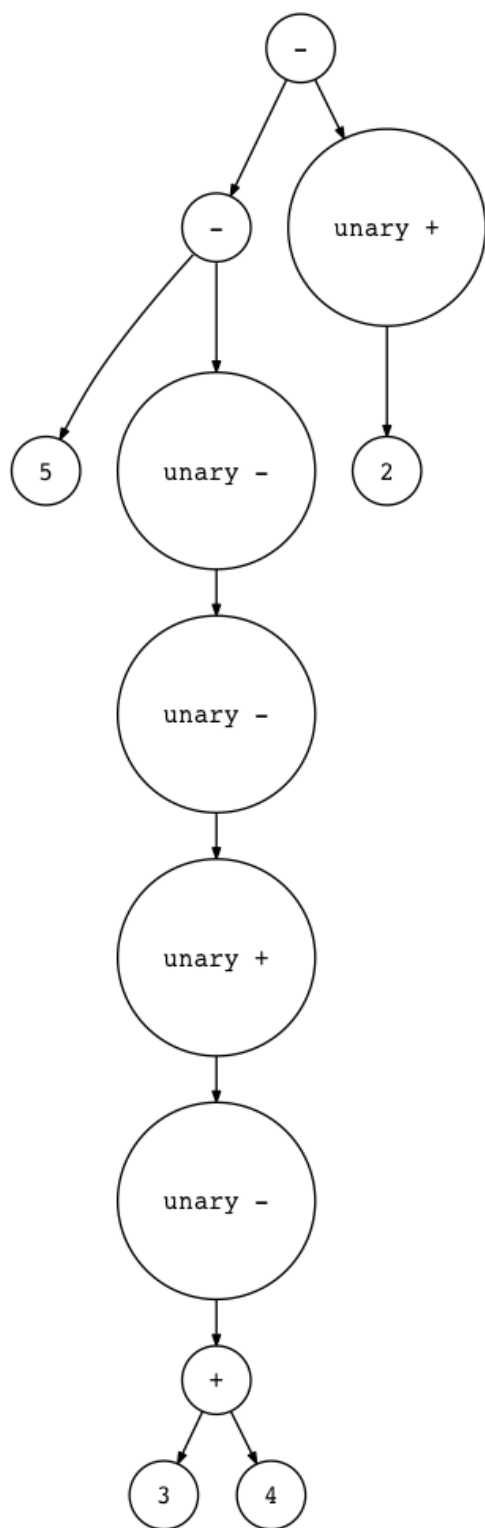
```
$ python genastdot.py "+ 3" > ast.dot && dot -Tpng -o ast.png ast.dot
```



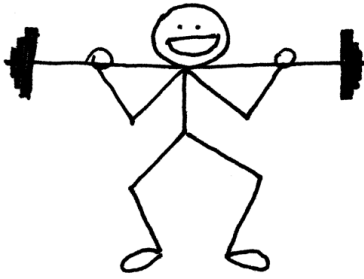
```
$ python genastdot.py "5 - - - + - 3" > ast.dot && dot -Tpng -o ast.png ast.dot
```



```
$ python genastdot.py "5 - - - + - (3 + 4) - +2" \  
> ast.dot && dot -Tpng -o ast.png ast.dot
```



这是给你的一个新练习：



- 安装Free Pascal (<http://www.freepascal.org/>), 编译并运行testunary.pas (<https://github.com/rspivak/lbasi/blob/master/part8/python/testunary.pas>), 并验证结果与使用spi (<https://github.com/rspivak/lbasi/blob/master/part8/python/spi.py>) 解释器生成的结果相同。

这就是今天的全部内容。在下一篇文章中, 我们将处理赋值语句。请继续关注, 很快就会见到你。

以下是我推荐的书籍清单, 它们将帮助您学习解释器和编译器:

1. 语言实现模式: 创建您自己的特定领域和通用编程语言 (实用程序员)
(http://www.amazon.com/gp/product/193435645X/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=193435645X&linkCode=as2&tag=russblo0b-20&linkId=MP4DCXDV6DJMEJBL)
2. 编写编译器和解释器: 一种软件工程方法
(http://www.amazon.com/gp/product/0470177071/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0470177071&linkCode=as2&tag=russblo0b-20&linkId=UCLGQTPIYSWYKRRM)
3. Java 中的现代编译器实现 (http://www.amazon.com/gp/product/052182060X/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=052182060X&linkCode=as2&tag=russblo0b-20&linkId=ZSKKZMV7YWR22NMW)
4. 现代编译器设计 (http://www.amazon.com/gp/product/1461446988/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=1461446988&linkCode=as2&tag=russblo0b-20&linkId=PAXWJP5WCPZ7RKRD)
5. 编译器: 原理、技术和工具 (第 2 版)
(http://www.amazon.com/gp/product/0321486811/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0321486811&linkCode=as2&tag=russblo0b-20&linkId=GOEGDQG4HIHU56FQ)

如果您想在收件箱中获取我的最新文章, 请在下方输入您的电子邮件地址, 然后单击“获取更新”!

输入您的名字 *

输入您最好的电子邮件 *

获取更新!

本系列所有文章:

- 让我们构建一个简单的解释器。第1部分。 (/lsbasi-part1/)
- 让我们构建一个简单的解释器。第2部分。 (/lsbasi-part2/)
- 让我们构建一个简单的解释器。第 3 部分。 (/lsbasi-part3/)
- 让我们构建一个简单的解释器。第 4 部分。 (/lsbasi-part4/)
- 让我们构建一个简单的解释器。第 5 部分。 (/lsbasi-part5/)
- 让我们构建一个简单的解释器。第 6 部分。 (/lsbasi-part6/)
- 让我们构建一个简单的解释器。第 7 部分: 抽象语法树 (/lsbasi-part7/)
- 让我们构建一个简单的解释器。第 8 部分。 (/lsbasi-part8/)
- 让我们构建一个简单的解释器。第 9 部分。 (/lsbasi-part9/)
- 让我们构建一个简单的解释器。第 10 部分。 (/lsbasi-part10/)
- 让我们构建一个简单的解释器。第 11 部分。 (/lsbasi-part11/)
- 让我们构建一个简单的解释器。第 12 部分。 (/lsbasi-part12/)
- 让我们构建一个简单的解释器。第 13 部分: 语义分析 (/lsbasi-part13/)
- 让我们构建一个简单的解释器。第 14 部分: 嵌套作用域和源到源编译器 (/lsbasi-part14/)
- 让我们构建一个简单的解释器。第 15 部分。 (/lsbasi-part15/)
- 让我们构建一个简单的解释器。第 16 部分: 识别过程调用 (/lsbasi-part16/)
- 让我们构建一个简单的解释器。第 17 部分: 调用堆栈和激活记录 (/lsbasi-part17/)
- 让我们构建一个简单的解释器。第 18 部分: 执行过程调用 (/lsbasi-part18/)
- 让我们构建一个简单的解释器。第 19 部分: 嵌套过程调用 (/lsbasi-part19/)

注释

ALSO ON RUSLAN'S BLOG

Let's Build A Simple Interpreter. Part 13: ...

4 years ago • 11 comments

Anything worth doing is worth overdoing. Before doing a deep dive into ...

Let's Build A Simple Interpreter. Part 15.

2 years ago • 14 comments

"I am a slow walker, but I never walk back." — Abraham Lincoln And ...

Let's Build A Simple Interpreter. Part 12.

5 years ago • 29 comments

"Be not afraid of going slowly; be afraid only of standing still." - Chinese ...

Let's B Interpr

2 years a

What I c not unde Richard

15 Comments

Ruslan's Blog

Disqus' Privacy Policy

1 Login

Recommend 6

Tweet

Share

Sort by Best

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

AJ • 5 years ago


Thank you for your work.

There is a mistake in typing "Let's manually build an AST for the expression "5 - - 2" ". You need to correct the following:

UnaryOp(minus_token, UnaryOp(minus_token, Num(two_tok)))

 [github \(https://github.com/rspivak/\)](https://github.com/rspivak/)

 [推特 \(https://twitter.com/rspivak\)](https://twitter.com/rspivak)

 [链接 \(https://linkedin.com/in/ruslanspivak/\)](https://linkedin.com/in/ruslanspivak/)

热门帖子

让我们构建一个 Web 服务器。第1部分。 (<https://ruslanspivak.com/lbaws-part1/>)

让我们构建一个简单的解释器。第1部分。 (<https://ruslanspivak.com/lbasi-part1/>)

让我们构建一个 Web 服务器。第2部分。 (<https://ruslanspivak.com/lbaws-part2/>)

让我们构建一个 Web 服务器。第 3 部分。 (<https://ruslanspivak.com/lbaws-part3/>)

让我们构建一个简单的解释器。第2部分。 (<https://ruslanspivak.com/lbasi-part2/>)

免责声明

这个网站上的一些链接有我的亚马逊推荐 ID，它为我提供了每次销售的小额佣金。感谢您的支持。