

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

日期 2015 年 7 月 3 日星期五

在他们的精彩著作《有效思维的 5 个要素》中，作者 Burger 和 Starbird 分享了一个故事，讲述了他们如何观察国际知名小号演奏家 Tony Plog 为有成就的小号演奏家举办大师班的故事。学生们首先演奏复杂的乐句，他们演奏得非常好。但随后他们被要求演奏非常基本、简单的音符。当他们演奏音符时，与之前演奏的复杂乐句相比，这些音符听起来很幼稚。弹完后，大师老师也弹了同样的音符，但弹奏起来，却一点儿也不幼稚。这种差异是惊人的。托尼解释说，掌握简单音符的演奏可以让人们以更好的控制演奏复杂的乐曲。教训很清楚——要建立真正的精湛技艺，必须专注于掌握简单的。<sup>1</sup>

故事中的教训显然不仅适用于音乐，也适用于软件开发。这个故事很好地提醒我们所有人不要忘记深入研究简单、基本的想法的重要性，即使有时感觉像是退步。虽然精通您使用的工具或框架很重要，但了解它们背后的原则也非常重要。正如拉尔夫沃尔多爱默生所说：

“如果你只学习方法，你就会被方法所束缚。但如果你学会了原则，你就可以设计出自己的方法。”

关于这一点，让我们再次深入研究解释器和编译器。

今天，我将向您展示第 1 部分中 (<http://ruslanspivak.com/lsbasi-part1/>) 计算器的新版本，它将能够：

1. 处理输入字符串中任意位置的空白字符
2. 从输入中使用多位整数
3. 两个整数相减（目前只能相加）

这是可以执行上述所有操作的新版本计算器的源代码：

```

# 标记类型
# EOF (end-of-file) 标记用于表示
# 没有更多的输入可供词法分析
INTEGER , PLUS , MINUS , EOF = 'INTEGER' , 'PLUS' , 'MINUS' , 'EOF '

class Token ( object ):
    def __init__ ( self , type , value ):
        # token type: INTEGER, PLUS, MINUS, or EOF
        self . type = type
        # 标记值: 非负整数值、'+'、'-' 或 None
        self . 价值 = 价值

    def __str__ ( self ):
        """类实例的字符串表示。

        实例:
            令牌 (INTEGER, 3)
            令牌 (PLUS '+')

        """
        返回 '令牌 ({类型}, {值})' 。 格式 (
            类型=自我。类型,
            值=再版 (自我。值)
        )

    def __repr__ ( self ):
        返回 self . __str__ ()

class Interpreter ( object ):
    def __init__ ( self , text ):
        # 客户端字符串输入, 例如"3 + 5"、"12 - 5"等
        self . text = text
        # self.pos 是 self.text
        self的索引。pos = 0
        # 当前令牌实例
        self . current_token = None
        self . current_char = self . 文本[自我。位置]

    def error ( self ):
        raise Exception ( 'Error parsing input' )

    DEF 提前 (自):
        """ , “提前‘POS’指针和设置‘current_char’变量”"""
        自我。pos += 1
        如果 self . pos > len ( self . text ) - 1 :
            self . current_char = None # 表示输入结束
        else :
            self . current_char = self . 文本[自我。位置]

    def skip_whitespace ( self ):
        while self . current_char 是 不 无 和 自我。current_char . isspace ():
            self . 提前()

    def integer ( self ):
        """返回从输入中消耗的 (多位) 整数。"""
        result = ''
        while self . current_char 是 不 无 和 自我。current_char . isdigit ():
            结果 += self . current_char
            自我。提前 ()
        返回 整数 (结果)

    def get_next_token ( self ):
        """词法分析器 (也称为扫描器或分词器)

```

此方法负责将句子  
分解为标记。

"""

而自我。current\_char 是不无:

```

    如果 自。current_char。isspace():
        self.skip_whitespace()
        继续

    如果 自。current_char。ISDIGIT():
        返回 令牌(INTEGER, 自我。整型())

    如果 自。current_char == '+':
        self.提前()
        返回 令牌(PLUS, '+')

    如果 自。current_char == '-':
        self.提前()
        返回 令牌(MINUS, '-')

    自我。错误()

    返回 令牌(EOF, 无)

```

```

def eat(self, token_type):
    # 比较当前标记类型与传递的标记
    # 类型, 如果它们匹配, 则“吃”当前标记
    # 并将下一个标记分配给 self.current_token,
    # 否则引发异常。
    如果 自。current_token.type == token_type:
        self.current_token = self.get_next_token()
    其他:
        自我。错误()

```

```

def expr(self):
    """解析器/解释器

    EXPR -> 整数加INTEGER
    EXPR -> INTEGER MINUS INTEGER
    """
    # 设定当前令牌从输入采取的第一令牌
    自。current_token = 自我.get_next_token()

    # 我们期望当前标记是一个整数
    left = self.current_token
    自我。吃(整数)

    # 我们期望当前的标记是 '+' 或 '-'
    op = self.current_token
    如果 op.类型 == PLUS:
        自我。吃(PLUS)
    其他:
        自我。吃(减)

    # 我们期望当前标记是一个整数
    right = self.current_token
    自我。吃(INTEGER)
    # 上述呼叫的self.current_token被设定为后
    # EOF标记

    # 在这一点上, INTEGER PLUS INTEGER 或
    # INTEGER MINUS INTEGER 标记序列
    # 已成功找到, 该方法可以
    # 返回两个整数的加减结果,
    # 从而有效地解释客户端输入
    if op.键入 == PLUS:

```

结果 = 左。价值 + 权利。其他值  
: 结果=左。价值-对。值返回结果

```
def main():
    while True:
        try:
            # 在 Python3 下运行替换 'raw_input' call
            # with 'input'
            text = raw_input('calc> ')
        except EOFError:
            break
        if not text:
            continue
        interpreter = Interpreter(text)
        result = 口译员.expr()
        打印(结果)

如果 __name__ == '__main__':
    main()
```

将上述代码保存到calc2.py文件中或直接从GitHub

(<https://github.com/rspivak/lbasi/blob/master/part2/calc2.py>)下载。试试看。亲眼看看它是否按预期工作：它可以处理输入中任何地方的空白字符；它可以接受多位整数，也可以减两个整数，也可以加两个整数。

这是我在笔记本电脑上运行的示例会话：

```
$ python calc2.py
计算> 27 + 3
30
计算> 27 - 7
20
计算>
```

与第 1 部分 (<http://ruslanspivak.com/lbasi-part1/>)的版本相比，主要的代码变化是：

1. 该get\_next\_token方法重构了一下。增加pos指针的逻辑被分解为一个单独的方法Advance。
2. 添加了另外两种方法：skip\_whitespace忽略空白字符和integer处理输入中的多位整数。
3. 该EXPR方法被修改，以识别INTEGER - > MINUS - > INTEGER短语除了INTEGER - > PLUS - > INTEGER短语。该方法现在还可以在成功识别相应短语后解释加法和减法。

在第 1 部分中，(<http://ruslanspivak.com/lbasi-part1/>)您学习了两个重要概念，即**标记**和**词法分析器**。今天我想谈谈**词素**、**解析**和**解析器**。

您已经了解令牌。但是为了让我完成对令牌的讨论，我需要提及词素。什么是词素？甲**词位**是形成一个令牌的字序列。在下图中，您可以看到一些标记和示例词素的示例，希望它可以使它们之间的关系变得清晰：

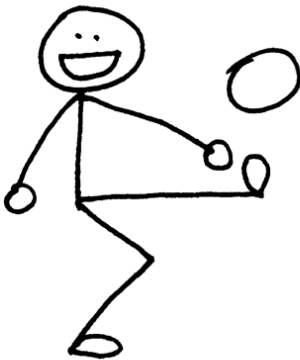
Token	Sample lexemes
INTEGER	342, 9, 0, 17, 1
PLUS	+
MINUS	-

现在，还记得我们的朋友expr方法吗？我之前说过，这就是算术表达式的解释实际发生的地方。但在解释一个表达式之前，您首先需要识别它是什么类型的短语，例如，是加法还是减法。这就是expr方法本质上所做的：它在从get\_next\_token方法获得的标记流中找到结构，然后解释已识别的短语，生成算术表达式的结果。

在标记流中找到结构的过程，或者换句话说，识别标记流中的短语的过程称为**解析**。执行该工作的解释器或编译器的部分称为**解析器**。

所以现在您知道expr方法是**解析**和**解释**发生的解释器的一部分- expr方法首先尝试识别（**解析**）INTEGER -> PLUS -> INTEGER或INTEGER -> MINUS -> INTEGER短语标记流，在成功识别（**解析**）其中一个短语后，该方法会对其进行解释，并将两个整数的加法或减法结果返回给调用者。

现在又到了锻炼的时候了。



1. 扩展计算器以处理两个整数的乘法
2. 扩展计算器以处理两个整数的除法
3. 修改代码以解释包含任意数量的加法和减法的表达式，例如 "9 - 5 + 3 + 11"

### 检查你的理解。

1. 什么是词素？
2. 在标记流中找到结构的过程的名称是什么，或者换句话说，识别该标记流中某个短语的过程的名称是什么？
3. 执行解析的解释器（编译器）部分的名称是什么？

我希望你喜欢今天的材料。在本系列的下一篇文章中，您将扩展计算器以处理更复杂的算术表达式。敬请关注。

这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-](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?](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)

[ie=UTF8&camp=1789&creative=9325&creativeASIN=0470177071&linkCode=as2&tag=russblo0b-20&linkId=UCLGQTPIYSWYKRRM\)](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?](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)

[ie=UTF8&camp=1789&creative=9325&creativeASIN=052182060X&linkCode=as2&tag=russblo0b-20&linkId=ZSKKZMV7YWR22NMW\)](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?](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)

[ie=UTF8&camp=1789&creative=9325&creativeASIN=1461446988&linkCode=as2&tag=russblo0b-20&linkId=PAXWJP5WCPZ7RKRD\)](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?](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)

[ie=UTF8&camp=1789&creative=9325&creativeASIN=0321486811&linkCode=as2&tag=russblo0b-20&linkId=GOEGDQG4HIHU56FQ\)](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 \(/lsbasi-part19/\)](#)

1. 有效思考的 5 个要素 ([http://www.amazon.com/gp/product/0691156662/ref=as\\_li\\_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0691156662&linkCode=as2&tag=russblo0b-20&linkId=B7GSVLONUPCIBIVY](http://www.amazon.com/gp/product/0691156662/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0691156662&linkCode=as2&tag=russblo0b-20&linkId=B7GSVLONUPCIBIVY)) ↩


## 注释

ALSO ON RUSLAN'S BLOG

<b>Let's Build A Simple Interpreter. Part 11.</b> 5 years ago • 18 comments I was sitting in my room the other day and thinking about how much we had ...	<b>Let's Build A Simple Interpreter. Part 19: ...</b> 2 years ago • 24 comments What I cannot create, I do not understand. — Richard Feynman	<b>Let's Build A Simple Interpreter. Part 16: ...</b> 2 years ago • 7 comments Learning is like rowing upstream: not to advance is to drop back. — Chinese ...	<b>Let's B Interpn</b> 6 years a Today w operator plus (+) :
---	--	--	--

16 Comments   Ruslan's Blog    Disqus' Privacy Policy

 Login ▾

 Recommend 6    Tweet    Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



**David Krupička** • 5 years ago • edited

Hi, thank you very much for the series!

For excersises, I have added support for multiple operators like this:

```
OPERATORS = {'+': lambda x,y: x + y,
'-': lambda x,y: x - y,
'*': lambda x,y: x * y,
'/': lambda x,y: x / y,
}
...
def get_next_token(self):
...
if self.current_char in OPERATORS:
token = Token(OPERATOR, self.current_char)
self.advance()
return token
...
def expr(self):
...
result = OPERATORS[op.value](left.value, right.value)
```

and replaced PLUS, MINUS etc. with OPERATOR

Nevertheless, I wonder about your solution in the next article! :-)

Many thanks.

^ | ▾ • Reply • Share ›



**Ritam Dey** → David Krupička • 4 years ago

You don't need to define those operation lambdas. Use the standard operators module

^ | ▾ • Reply • Share ›



**Дмитрий Голубков** • 5 years ago



Could you explain me, why do you using while in `get_next_token`?  
We could use another if and return EOF if char is None

^ | v • Reply • Share ›



**Rahul Sharma** • 5 years ago • edited

For skipping whitespace, we can rely on the recursive nature of reading input and just modify `get_next_token()` by adding just three lines:

```
def get_next_token(self):
    .....
    ....
    # if we see a space we just increment the pos, and recursively call get_next_token
    if current_char.isspace():
        self.pos+=1
    return self.get_next_token()
    .....
    .....
    self.error()
```

^ | v • Reply • Share ›



**lucid** • 5 years ago

Thanks alot for this series, it's really great and detailed, easy to follow. Really nice work!

^ | v • Reply • Share ›



**Алексей Магдич** • 5 years ago

It's great! Thanks! But i have a simple question: why did we write "`skip_whitespace()`" function, if we can just add, for example, in constructor of our Interpreter such string as "`self.text = text.replace(" ", "")`" and all whitespaces will dissapear at one moment?

^ | v • Reply • Share ›



**Igor Pantović** → Алексей Магдич • 5 years ago

Because we can't just skip any whitespace. If we did, this would happen:

`12 + 2 => 12+2 --- All good`

`1 45 23 + 2 => 14523+2 ---- This wouldn't be right, it should throw syntax error instead of calculating`

^ | v • Reply • Share ›



**Shubham Sinha** → Igor Pantović • 4 years ago

why it will not be calculated?

^ | v • Reply • Share ›



**Anonym** • 6 years ago

Again, very nicely written.

Here's a Java version of your code to play with:

<https://codeboard.io/projec...>

And my personal solution to the exercise:

<https://codeboard.io/projec...>

Maybe someone finds it helpful.

^ | v • Reply • Share ›



**thulani mtetwa** → Anonym • 5 years ago

Great work on the java version of this again

^ | v • Reply • Share ›



**Andrew** • 6 years ago

First of all congratulations for your work, very useful. Have you read "Principles and practice using C++" by Stroustrup? In the first chapters he makes a very useful description about the construction of a calculator. I think that book can be a great additional resource for the part of `token_parser` and so on



token, parser and so on.

^ | v • Reply • Share ›



**Kir** • 6 years ago

I would be pleased, if you add "Compiling with continuations" to the list of books. It's a great one, too.

^ | v • Reply • Share ›



**Paul** • 6 years ago

This is really good stuff. I hope you keep doing these. I've learned a lot from all your posts so far.

^ | v • Reply • Share ›



**rspivak** Mod → **Paul** • 6 years ago

Thanks, Paul.

^ | v • Reply • Share ›



**Anonym** • 6 years ago

Excellent! Thanks a lot for part 2.

^ | v • Reply • Share ›



**rspivak** Mod → **Anonym** • 6 years ago

You're welcome :)

## 🏠 社会的

🐙 github (<https://github.com/rspivak/>)

🐦 推特 (<https://twitter.com/rspivak>)

🌐 链接 (<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。