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

日期 2015 年 6 月 15 日，星期一

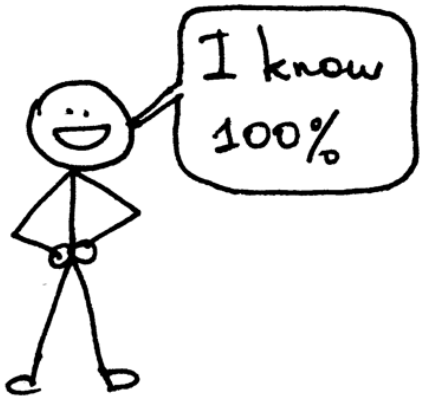
> **“如果你不知道编译器是如何工作的，那么你就不知道计算机是如何工作的。如果你不能 100% 确定你是否知道编译器是如何工作的，那么你就不知道它们是如何工作的。”** — 史蒂夫·耶格

你有它。想想看。无论您是新手还是经验丰富的软件开发人员都没有关系：如果您不知道编译器和解释器的工作原理，那么您就不知道计算机的工作原理。就这么简单。

那么，你知道编译器和解释器是如何工作的吗？我的意思是，您是否 100% 确定您知道它们是如何工作的？如果你不这样做。

或者，如果您不这样做并且您真的对此感到不安。

不要担心。如果你坚持并完成这个系列并和我一起构建一个解释器和一个编译器，你最终会知道它们是如何工作的。你也将成为一个自信的快乐露营者。至少我希望如此。



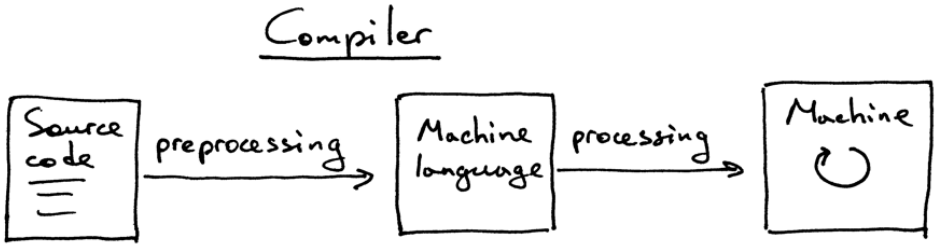
你为什么要学习解释器和编译器？我给你三个理由。

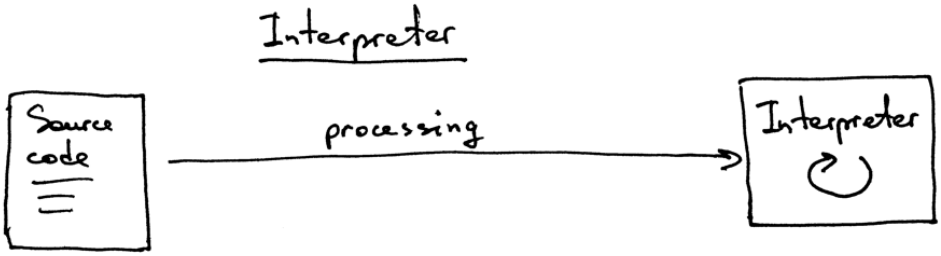
1. 要编写解释器或编译器，您必须具备许多需要一起使用的技术技能。编写解释器或编译器将帮助您提高这些技能并成为更好的软件开发人员。同样，您将学到的技能在编写任何软件时都非常有用，而不仅仅是解释器或编译器。
2. 你真的很想知道计算机是如何工作的。通常解释器和编译器看起来很神奇。你不应该对这种魔法感到满意。您想揭开构建解释器和编译器的过程的神秘面纱，了解它们的工作原理并控制事物。
3. 您想创建自己的编程语言或领域特定语言。如果你创建了一个，你还需要为它创建一个解释器或编译器。最近，人们对新的编程语言重新产生了兴趣。你几乎每天都能看到一种新的编程语言出现：Elixir、Go、Rust 等等。

好的，但是什么是解释器和编译器？

解释器或**编译器**的目标是将某种高级语言的源程序翻译成某种其他形式。很模糊，不是吗？请耐心等待，在本系列的后面，您将准确了解源程序被翻译成什么。

此时您可能还想知道解释器和编译器之间的区别是什么。出于本系列的目的，我们同意，如果翻译器将源程序翻译成机器语言，那么它就是**编译器**。如果翻译器处理和执行源程序而不先将其翻译成机器语言，那么它就是**解释器**。从视觉上看，它看起来像这样：





我希望现在你确信你真的想研究和构建一个解释器和一个编译器。你对这个关于口译员的系列有什么期待？

这是交易。你和我将为Pascal (https://en.wikipedia.org/wiki/Pascal_%28programming_language%29)语言的一大子集创建一个简单的解释器。在本系列结束时，您将拥有一个可工作的 Pascal 解释器和一个源代码级调试器，如 Python 的pdb (https://docs.python.org/2/library/pdb.html)。

你可能会问，为什么是帕斯卡？一方面，它不是我为这个系列专门设计的一种编造语言：它是一种真正的编程语言，具有许多重要的语言结构。一些古老但有用的CS书籍在他们的示例中使用 Pascal 编程语言（我知道这不是选择一种语言来构建解释器的特别令人信服的理由，但我认为改变学习非- 主流语言 :)

以下是 Pascal 中阶乘函数的示例，您将能够使用您自己的解释器进行解释，并使用您将在此过程中创建的交互式源级调试器进行调试：

```
程序 阶乘;

函数 阶乘（n : 整数）: longint ;
如果n = 0则开始
    阶乘:= 1否则阶乘:= n *阶乘( n - 1 ) ; 结束;




var
    n : 整数;

begin
    for  n  :=  0  to  16  do
        writeln ( n ,  '!=' ,  factorial ( n )) ;
结束。
```

Pascal 解释器的实现语言将是 Python，但您可以使用任何您想要的语言，因为所呈现的思想不依赖于任何特定的实现语言。好的，让我们进入正题。预备，准备，开始！

您将通过编写一个简单的算术表达式解释器（也称为计算器）来开始您对解释器和编译器的首次尝试。今天的目标是非常简单的：让你的计算器处理两个单数整数的加法，比如**3+5**。这是您的计算器的源代码，抱歉，解释器：

```
# 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 , EOF = 'INTEGER' , 'PLUS' , 'EOF'


class Token ( object ):
    def __init__ ( self , type , value ):
        # token type: INTEGER, PLUS, or EOF
        self . type = type
        # token value: 0, 1, 2. 3, 4, 5, 6, 7, 8, 9, '+', or 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"
        self 。text = text
        # self.pos 是 self.text
        self的索引。pos = 0
        # 当前令牌实例
        self . current_token = 无

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

    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。一次一个令牌。
        """
        text = self . text

        # self.pos 索引是否超过了 self.text 的结尾？
        # 如果是这样，则返回 EOF 标记，因为没有更多的
        输入要转换为标记
        if self 。pos > len ( text ) - 1 :
            return Token ( EOF , None )

        # 在 self.pos 位置获取一个字符并决定
        # 根据单个字符创建什么标记
        current_char = text [ self . 位置]

        # 如果字符是数字，则将其转换为
        # 整数，创建一个整数标记，增加 self.pos
        # index 以指向数字后的下一个字符，
        # 并返回整数标记
        if current_char 。isdigit ():
```

```
            token  =  Token ( INTEGER ,  int ( current_char ))
            self 。pos  +=  1
            返回 令牌

        if  current_char  ==  '+' :
            token  =  Token ( PLUS ,  current_char )
            self 。pos  +=  1
            返回 令牌

        自我。错误()

    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 ):
        """expr -> INTEGER PLUS INTEGER"""
        # 将当前标记设置为从输入
        self 中获取的第一个标记。current_token  =  self 。get_next_token ()

        # 我们期望当前的标记是一个个位数的整数
        left  =  self . current_token
        自我。吃（整数）

        # 我们期望当前标记是一个 '+' 标记
        op  =  self . current_token
        自我。吃（PLUS ）

        # 我们期望当前的标记是一个个位数的整数
        right  =  self . current_token
        自我。吃（INTEGER ）
        #上述呼叫的self.current_token被设定为后
        #EOF标记

        #此时INTEGER PLUS INTEGER令牌序列
        #已成功找到，该方法只需
        #返回两个整数相加的结果，从而
        #有效解释客户端输入
        result  =  left 。价值 + 权利。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calc1.py文件中或直接从GitHub (https://github.com/rspivak/lsbasi/blob/master/part1/calc1.py)下载。在开始深入研究代码之前，请在命令行上运行计算器并查看它的运行情况。玩它！这是我的笔记本电脑上的示例会话（如果您想在 Python3 下运行计算器，您需要将raw_input替换为input）：

```
$ python calc1.py
计算> 3 +4
 7
计算> 3 +5
 8
计算> 3 +9
 12
计算>
```

为了让您的简单计算器正常工作而不抛出异常，您的输入需要遵循某些规则：

- 输入中只允许单个数字整数
- 目前唯一支持的算术运算是加法
- 输入中的任何地方都不允许有空格字符

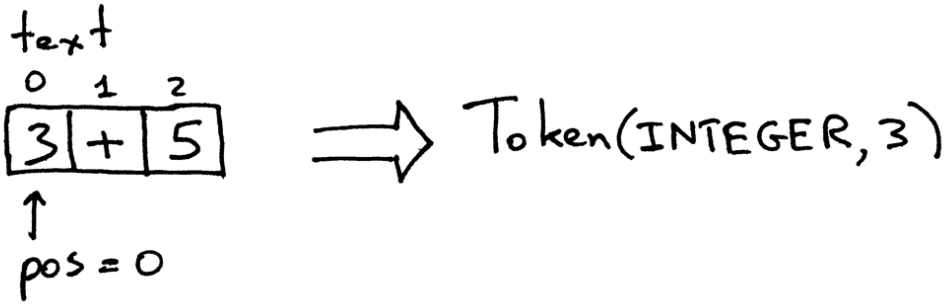
这些限制是使计算器简单所必需的。别担心，你很快就会让它变得非常复杂。

好的，现在让我们深入了解一下你的解释器是如何工作的，以及它是如何计算算术表达式的。

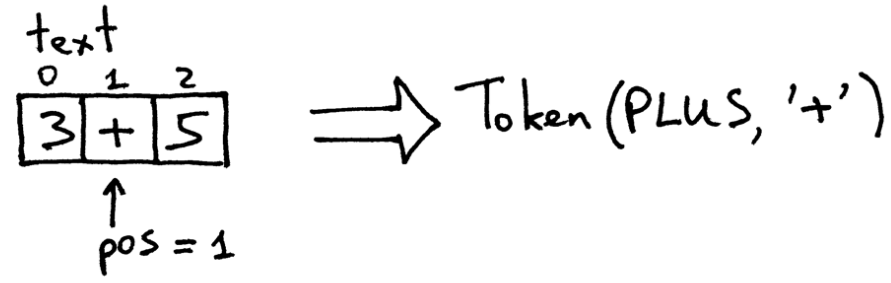
当您在命令行中输入表达式3+5时，您的解释器会得到一个字符串“3+5”。为了让解释器真正理解如何处理该字符串，它首先需要将输入“3+5”分解为称为**tokens 的**组件。甲**令牌**是具有类型和一个值的对象。例如，对于字符串“3”，标记的类型将为INTEGER，对应的值为整数3。

将输入字符串分解为标记的过程称为**词法分析**。因此，您的解释器需要做的第一步是读取字符输入并将其转换为令牌流。其做它的解释器的部分被称为一个**词法分析器**，或**词法分析器**的简称。您可能还会遇到同一组件的其他名称，例如**扫描器**或**标记器**。它们的意思都是一样的：解释器或编译器的一部分，将字符输入转换为标记流。

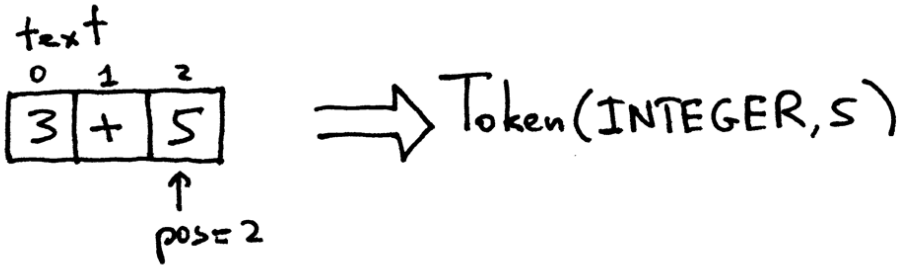
该方法get_next_token的的解释类是你的词法分析器。每次调用它时，都会从传递给解释器的字符输入中获得下一个令牌。让我们仔细看看这个方法本身，看看它实际上是如何完成将字符转换为标记的工作的。输入存储在保存输入字符串的变量text 中，pos是该字符串的索引（将字符串视为字符数组）。pos最初设置为 0 并指向字符'3'。该方法首先检查字符是否为数字，如果是，则增加pos并返回具有类型的标记实例INTEGER和设置为字符串'3'的整数值的值，它是一个整数3：



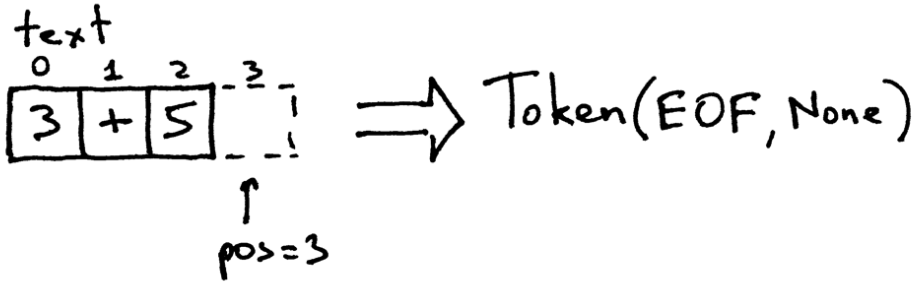
该POS现在指向“+”在字符的文本。下次调用该方法时，它会测试位置pos处的字符是否为数字，然后测试该字符是否为加号，它是。因此，该方法增加pos并返回一个新创建的类型为PLUS且值为'+' 的令牌：



该POS现在指向字符“5”。当您再次调用get_next_token方法时，该方法会检查它是否是数字，因此它会增加pos并返回一个新的INTEGER令牌，该令牌的值设置为整数5：



因为pos索引现在超过了字符串“3+5”的末尾，所以每次调用get_next_token方法时都会返回EOF标记：



尝试一下，亲眼看看计算器的词法分析器组件是如何工作的：

```
>>> from calc1 import Interpreter
>>>
>>> interpreter = Interpreter ( '3+5' )
>>> interpreter 。get_next_token ()
令牌( INTEGER , 3 )
>>>
>>> 解释器。get_next_token ()
令牌（PLUS , '+' ）
>>>
>>> 解释器。get_next_token ()
令牌（整数，

>>> 翻译。get_next_token ()
令牌( EOF , None )
>>>
```

所以现在你的解释器可以访问由输入字符组成的标记流，解释器需要对它做一些事情：它需要在它从词法分析器get_next_token获得的标记流中找到结构。您的解释器希望在该流中找到以下结构：INTEGER -> PLUS -> INTEGER。也就是说，它试图找到一个标记序列：整数后跟一个加号后跟一个整数。

负责查找和解释该结构的方法是expr。此方法验证令牌序列确实对应于预期的令牌序列，即INTEGER -> PLUS -> INTEGER。它的成功确认结构后，它产生加在左侧的标记的值结果PLUS和右侧PLUS，从而成功地解释你传递给解释算术表达式。

该EXPR方法本身使用的辅助方法，吃来验证传递给令牌类型吃法当前标记类型相匹配。在匹配传递的标记类型后，eat方法获取下一个标记并将其分配给current_token变量，从而有效地“吃掉”当前匹配的标记并推进标记流中的虚指针。如果令牌流中的结构与预期的INTEGER PLUS INTEGER令牌序列不对应，eat方法将引发异常。

让我们回顾一下您的解释器如何评估算术表达式：

- 解释器接受一个输入字符串，比如说“3+5”
- 解释器调用expr方法在词法分析器get_next_token返回的标记流中查找结构。它试图找到的结构是INTEGER PLUS INTEGER的形式。在确认结构后，它通过添加两个INTEGER标记的值来解释输入，因为此时解释器很清楚它需要做的是添加两个整数 3 和 5。

祝贺你自己。您刚刚学会了如何构建您的第一个解释器！

现在是练习的时候了。

你不认为你会读这篇文章就足够了，是吗？好的，把手弄脏并做以下练习：

1. 修改代码以允许输入中的多位整数，例如“12+3”
2. 添加一种跳过空白字符的方法，以便您的计算器可以处理带有空白字符的输入，例如“12 + 3”
3. 修改代码，而不是 '+' 句柄 '-' 来计算像“7-5”这样的减法

检查你的理解

1. 什么是口译员？
2. 什么是编译器？
3. 解释器和编译器有什么区别？
4. 什么是令牌？
5. 将输入分解为标记的过程的名称是什么？
6. 进行词法分析的解释器的部分是什么？
7. 解释器或编译器的那部分的其他常见名称是什么？

在我完成这篇文章之前，我真的希望你致力于研究解释器和编译器。我希望你现在就去做。不要把它放在次要位置。别等了。如果您浏览了文章，请重新开始。如果你已经仔细阅读过但还没有做过练习——现在就去做吧。如果您只完成了其中的一部分，请完成其余部分。你明白了。你知道吗？签署承诺，从今天开始学习解释器和编译器！

我，_ _ _ _ ____，身心健全，特此承诺从今天开始致力于研究解释器和编译器，并达到我 100% 了解它们如何工作的地步！

签名：

日期：

签名，注明日期，并将其放在您每天都能看到的地方，以确保您恪守承诺。并记住承诺的定义：

> “承诺是在你说它的情绪离开你很久之后做你说你会做的事情。” — 达伦·哈迪

好了，今天就到此为止。在迷你系列的下一篇文章中，您将扩展您的计算器以处理更多算术表达式。敬请关注。

如果您迫不及待地等待第二篇文章，并且正在绞尽脑汁开始深入研究解释器和编译器，这里是我推荐的书籍清单，它们将帮助您一路走来：

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 3.

6 years ago • 15 comments

I woke up this morning and I thought to myself: "Why do we find it so difficult to …

Let's Build A Simple Interpreter. Part 7: …

6 years ago • 23 comments

As I promised you last time, today I will talk about one of the central data …

Let's Build A Web Server. Part 1.

7 years ago • 88 comments

Out for a walk one day, a woman came across a construction site and saw …

Let's B Interpr

6 years a

Have yo learning these ar

61 Comments Ruslan's Blog Disqus' Privacy Policy Login

Recommend 31 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS

Name

Jack Crenshaw • 5 years ago

I enjoyed reading your tutorial, but I do sorta wish you'd referenced my own tutorial, "Let's Build a Compiler,"

http://compilers.iecc.com/c...
which I started in 1988. My version has been in the public domain since then, and has been ported to virtually every target machine and every programming language anyone could want.

If nothing else, I'm thinking maybe I should have Copyrighted the "Let's Build a ..." tag line ;-)

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/lsbaws-part1/)
- 让我们构建一个简单的解释器。第1部分。 (https://ruslanspivak.com/lsbasi-part1/)
- 让我们构建一个 Web 服务器。第2部分。 (https://ruslanspivak.com/lsbaws-part2/)
- 让我们构建一个 Web 服务器。第 3 部分。 (https://ruslanspivak.com/lsbaws-part3/)
- 让我们构建一个简单的解释器。第2部分。 (https://ruslanspivak.com/lsbasi-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。

回到顶部