

让我们构建一个简单的解释器。第 19 部分：嵌套过程调用 (<https://ruslanspivak.com/lsbasi-part19/>)

日期 2020 年 3 月 19 日, 星期四

“我无法创造的东西，我不明白。” — 理查德·费曼

正如我上次向您保证的那样，今天我们将扩展上一篇文章中涵盖的内容，并讨论执行嵌套过程调用。就像上次一样，我们今天将重点限制在只能访问其参数和局部变量的过程上。我们将在下一篇文章中介绍访问非局部变量。这是今天的示例程序：

```
程序 主程序；

程序 Alpha (a : 整数; b : 整数);
var x : 整数;

    程序 Beta (a : 整数; b : 整数);
    var x : 整数;
    开始
        x := a * 10 + b * 2 ;
    结束;

开始
    x := (a + b) * 2 ;

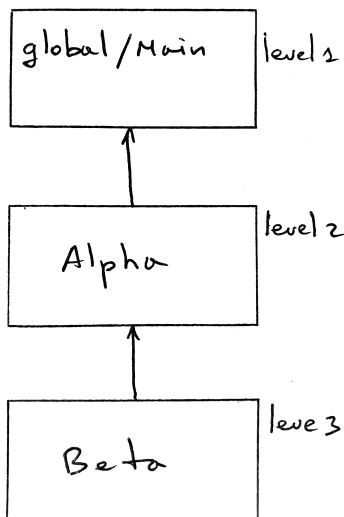
    测试版( 5 , 10 );      {过程调用}
结束;

开始 {主要}

    阿尔法( 3 + 5 , 7 );   { 过程调用 }

结束。 { 主要的 }
```

程序的嵌套关系图如下所示：



关于上述程序的一些注意事项：

- 它有两个过程声明，Alpha和Beta
- Alpha在主程序（全局 范围）中声明
- 该测试版程序的内部声明阿尔法 程序
- 两者的Alpha和Beta版有他们的形式参数相同的名字：整数一和b
- 并且Alpha和Beta具有相同的局部变量x
- 程序有嵌套调用：Beta过程是从Alpha过程中调用的，而Alpha过程又是从主程序中调用的

现在，让我们做一个实验。从GitHub (<https://github.com/rspivak/lbasi/blob/master/part19>)下载 part19.pas (<https://github.com/rspivak/lbasi/blob/master/part19/part19.pas>)文件，并以part19.pas (<https://github.com/rspivak/lbasi/blob/master/part19/part19.pas>)文件作为输入运行上一篇文章中 (<https://github.com/rspivak/lbasi/blob/master/part18/spi.py>)的解释器 (<https://github.com/rspivak/lbasi/blob/master/part18/spi.py>)，看看当解释器执行嵌套过程调用（主程序调用Alpha调用Beta）时会发生什么： (<https://github.com/rspivak/lbasi/blob/master/part19>) (<https://github.com/rspivak/lbasi/blob/master/part18/spi.py>) (<https://github.com/rspivak/lbasi/blob/master/part19/part19.pas>)

```
$ python spi.py part19.pas --stack
```

输入：程序主

调用堆栈

1 : 程序主

...

输入：程序测试版

调用堆栈

2: 程序测试版

a : 5

b : 10

2 : 程序阿尔法

a : 8

b : 7

x : 30

1 : PROGRAM Main

休假：程序测试版

调用堆栈

2: 程序测试版

a : 5

b : 10

x : 70

2 : 程序阿尔法

a : 8

b : 7

x : 30

1 : PROGRAM Main

...

离开：程序主要

调用堆栈

1 : 程序主

它只是有效！没有错误。如果您研究 AR（激活记录）的内容，您会发现Alpha和Beta过程调用的激活记录中存储的值是正确的。那么，有什么收获呢？有一个小问题。如果您查看显示“ENTER：PROCEDURE Beta”的输出，您会发现Beta和Alpha过程调用的嵌套级别相同，为 2（二）。Alpha的嵌套级别应该是 2，Beta的嵌套级别应该是 3。这是我们需要解决的问题。现在的nesting_level值visit_ProcedureCall方法被硬编码为 2（二）：

```
def visit_ProcedureCall ( self , node ):
    proc_name = node 。 进程名称

    ar = ActivationRecord (
        name = proc_name ,
        type = ARType . PROCEDURE ,
        nesting_level = 2 ,
    )

    proc_symbol = 节点。proc_symbol
    ...
```

让我们摆脱硬编码值。我们如何确定过程调用的嵌套级别？在上面的方法中，我们有一个过程符号，它存储在具有正确范围级别的范围符号表中，我们可以将其用作nesting_level参数的值（有关范围和范围级别的更多详细信息，请参阅第 14 部分 (<https://ruslanspivak.com/lbasi-part14/>)）。

我们如何通过过程符号到达作用域符号表的作用域级别？

让我们看看ScopedSymbolTable 类的以下部分：

```
类 ScopedSymbolTable :
    def __init__ ( self , scope_name , scope_level , enclosure_scope = None ):
        ...
        self 。 scope_level = scope_level
        ...

    def 插入 ( 自我 , 符号 ):
        自我。log ( f '插入: {symbol.name}' )
        self 。 _symbols [ 符号。名称 ] = 符号
```

看上面的代码，我们可以看到，当我们在insert方法内部的作用域符号表（scope）中存储符号时，我们可以将一个作用域符号表的作用域级别分配给一个符号。这样我们就可以在解释阶段访问visit_Procedure方法中的过程符号的范围级别。而这正是我们所需要的。

让我们进行必要的更改：

- 首先，让我们向Symbol类添加一个scope_level成员，并给它一个默认值零：

```
类 符号:
    def __init__ ( self , name , type = None ):
        ...
        self 。 范围_级别 = 0
```

- 接下来，让我们在将符号存储在作用域符号表中时，为该符号分配相应的作用域级别：

```
class ScopedSymbolTable :
    ...
    def insert ( self , symbol ):
        self 。 日志 ( f '插入: {symbol.name}' )

        符号。scope_level = self 。 范围_级别

        自我。_symbols [ 符号。名称 ] = 符号
```

现在，当在visit_ProcedureCall方法中为过程调用创建AR时，我们可以访问过程符号的范围级别。剩下要做的就是使用过程符号的范围级别作为nesting_level 参数的值：

```
class Interpreter ( NodeVisitor ):  
    ...  
    def visit_ProcedureCall ( self , node ):  
        proc_name = node . proc_name  
        proc_symbol = 节点 . proc_symbol  
  
        AR = ActivationRecord (   
            名称= proc_name中 ,  
            类型= ARType . PROCEDURE ,  
            nesting_level = proc_symbol . scope_level + 1 ,  
        )
```

太好了，没有更多的硬编码嵌套级别。值得一提的是，为什么我们将proc_symbol.scope_level + 1作为nesting_level参数的值，而不仅仅是proc_symbol.scope_level。在第 17 部分中 (<https://ruslanspivak.com/lbasi-part17/>)，我提到AR的嵌套级别对应于相应过程或函数声明的范围级别加一。让我们看看为什么。

在我们今天的示例程序中，Alpha过程符号 - 包含有关Alpha过程声明的信息的符号 - 存储在级别 1（一）的全局作用域中。所以 1 是Alpha过程符号的scope_level 的值。但是正如我们从Part14 中 (<https://ruslanspivak.com/lbasi-part14/>)知道的，过程声明Alpha的作用域级别比在过程Alpha内部声明的变量的级别低一级。因此，要获得存储Alpha过程的参数和局部变量的作用域的作用域级别，我们需要将过程符号的作用域级别增加 1。这就是我们使用的原因proc_symbol.scope_level + 1作为为过程调用创建AR时nesting_level参数的值，而不仅仅是proc_symbol.scope_level。

让我们看看到目前为止我们所做的更改。下载[更新的解释器](#) (<https://github.com/rspivak/lbasi/blob/master/part19>)并使用part19.pas (<https://github.com/rspivak/lbasi/blob/master/part19>)文件作为其输入再次测试：

```
$ python spi.py part19.pas --stack
```

输入：程序主

调用堆栈

1 : 程序主

输入：程序阿尔法

调用堆栈

2: 程序阿尔法

a : 8

b : 7

1 : PROGRAM Main

输入：程序测试版

调用堆栈

3: 程序测试版

a : 5

b : 10

2 : 程序阿尔法

a : 8

b : 7

x : 30

1 : PROGRAM Main

休假：程序测试版

调用堆栈

3: 程序测试版

a : 5

b : 10

x : 70

2 : 程序阿尔法

a : 8

b : 7

x : 30

1 : PROGRAM Main

离开：程序阿尔法

调用堆栈

2: 程序阿尔法

a : 8

b : 7

x : 30

1 : PROGRAM Main

离开：程序主要

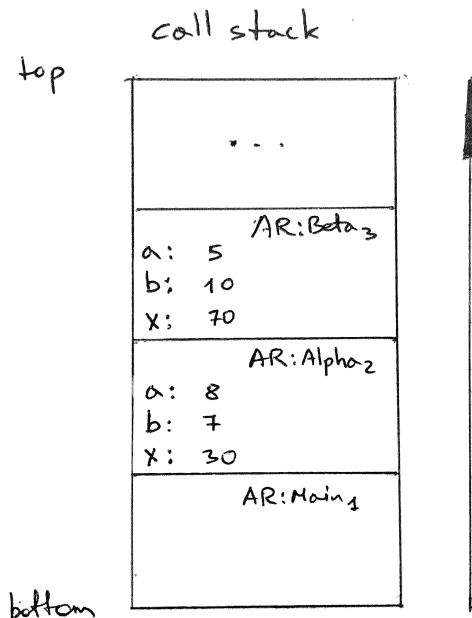
调用堆栈

1 : 程序主

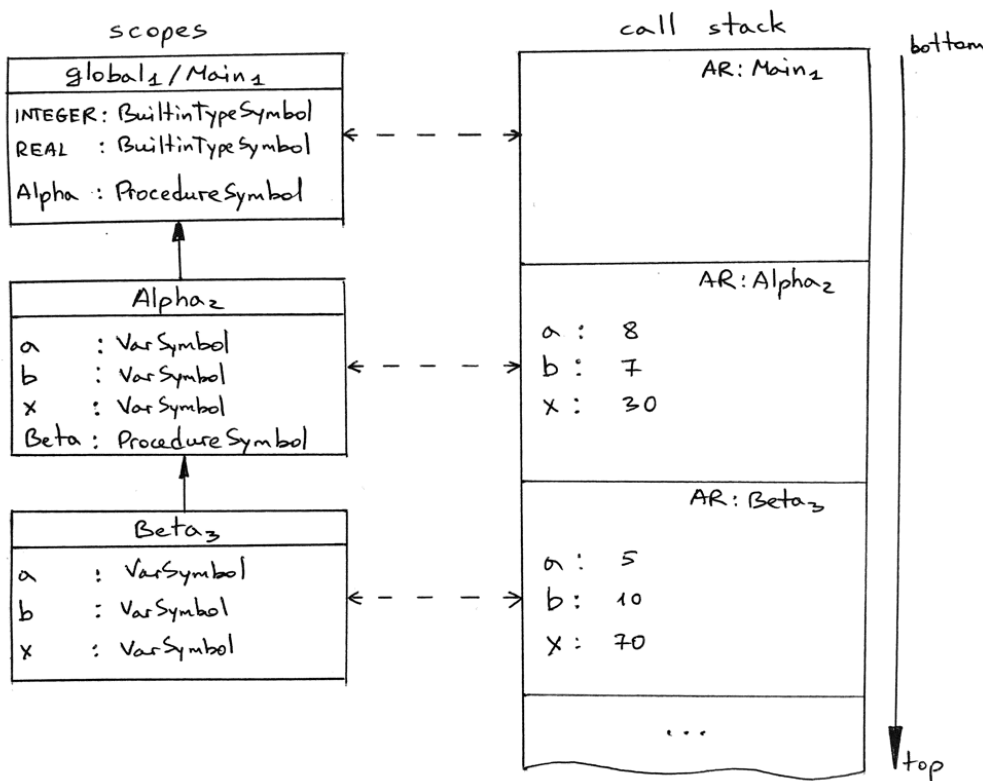
从上面的输出中可以看出，活动记录 (AR)的嵌套级别现在具有正确的值：

- 该主要程序AR：嵌套级1
- 在阿尔法过程AR：嵌套级2
- 该测试版程序AR：嵌套3级

让我们来看看在程序执行过程中作用域树（作用域符号表）和调用堆栈是如何视觉化的。这里是调用堆栈看起来多么消息之后 “ LEAVE：程序测试版” 与之前AR的测试版程序调用出栈：



如果我们翻转调用堆栈（使堆栈顶部位于“底部”），您可以看到带有活动记录的调用堆栈如何与带有作用域的作用域树（作用域符号表）相关联。事实上，我们可以说活动记录是作用域的运行时等价物。作用域是在源程序的语义分析期间创建的（在此阶段读取、解析和分析源程序，但不执行），并且在解释器执行源程序时在运行时创建带有活动记录的调用堆栈：



正如您在本文中看到的，我们没有进行大量更改来支持嵌套过程调用的执行。唯一真正的变化是确保 AR 中的嵌套级别是正确的。代码库的其余部分保持不变。我们的代码在嵌套过程调用中几乎不变地继续工作的主要原因是示例程序中的 Alpha 和 Beta 过程仅访问局部变量的值（包括它们自己的参数）。并且因为这些值存储在堆栈顶部的 AR 中，这允许我们继续使用方法 visit_Assignment 和 visit_Var 在执行程序主体时，无需任何更改。下面是这些方法的源代码：

```
def visit_Assign ( self , node ):  
    var_name = node 。离开了。值  
    var_value = self 。访问 (节点。右)  
  
    ar = 自我。调用堆栈。peek ()  
    ar [ var_name ] = var_value  
  
def visit_Var ( self , node ):  
    var_name = node 。价值  
  
    ar = 自我。调用堆栈。peek ()  
    var_value = ar 。获取 ( var_name )  
  
    返回 变量值
```

好的，今天我们已经能够使用我们的解释器成功执行嵌套过程调用，只需很少的更改。现在我们离正确执行递归过程调用又近了一步。

这就是今天的内容。在下一篇文章中，我们将讨论过程如何在运行时访问非局部变量。

保持安全，保持健康，互相照顾！下次见。

用于准备本文的资源（链接是附属链接）：

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）
(https://www.amazon.com/gp/product/193435645X/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=193435645X&linkCode=as2&tag=russblo0b-20&linkId=5d5ca8c07bff5452ea443d8319e7703d)
2. 编写编译器和解释器：一种软件工程方法
(https://www.amazon.com/gp/product/0470177071/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0470177071&linkCode=as2&tag=russblo0b-20&linkId=542d1267e34a529e0f69027af20e27f3)
3. 编程语言语用学，第四版 (https://www.amazon.com/gp/product/0124104096/ref=as_li_tl?ie=UTF8&camp=1789&creative=9325&creativeASIN=0124104096&linkCode=as2&tag=russblo0b-20&linkId=8db1da254b12fe6da1379957dda717fc)

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

<p>Let's Build A Simple Interpreter. Part 3.</p> <p>6 years ago • 15 comments</p> <p>I woke up this morning and I thought to myself: "Why do we find it so difficult to ..."</p>	<p>Let's Build A Simple Interpreter. Part 11.</p> <p>5 years ago • 18 comments</p> <p>I was sitting in my room the other day and thinking about how much we had ...</p>	<p>Let's Build A Simple Interpreter. Part 17: ...</p> <p>2 years ago • 8 comments</p> <p>You may have to fight a battle more than once to win it. - Margaret Thatcher</p>	<p>Let's Build A Simple Interpreter. Part 9.</p> <p>5 years ago • 42 comments</p> <p>I remember when I was in university (a long time ago) and learning systems ...</p>	<p>Let's Interj</p> <p>4 years</p> <p>Only d flow. A last ar</p>
---	--	--	--	---

24 Comments Ruslan's Blog  Disqus' Privacy Policy

 Login ▾

 Recommend 4  Tweet  Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS 

Name



The Stranger • 5 months ago

Hi,
I'm trying to build this interpreter from scratch without classes.
But I couldn't retype a function based visitor.
I am dead stuck at the moment.
Is there any idea for it?

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

1 ^ | ▾ • Reply • Share ›



jfox • 5 months ago

Hey Ruslan! Just finished "Let's Build A Simple Interpreter. Part 19". It's been a great series but I must confess I am disappointed that you never finished it. I was really looking forward to getting into functions, decisions, etc. I hope everything is okay with you.

1 ^ | ▾ • Reply • Share ›

**Irwin Rodriguez** • 4 months agoMe too, but I couldn't wait for the next chapter and took my own way: <https://github.com/Irwin198...>

^ | v • Reply • Share ›

**rspivak** Mod • 4 months ago

I'm doing all right, thank you. Glad you liked the series and I do hope to pick it up where I left off at some point in the future.

^ | v • Reply • Share ›

**Irwin Rodriguez** • 8 months agoI'm finally here! and this is the proof: <https://github.com/Irwin198...>

Thanks for every single word you typed here Ruslan. It wasn't easy for you to read, understand, resume and write here all you had to understand by reading tones of books about compilers design. I appreciate that effort from your part. Now while you prepare the next chapter I'll be trying to continue with nested procedure calls and add support for functions and return statement. Thanks again and stay safe!

1 ^ | v • Reply • Share ›

**liang zhang** • 9 months ago

I implement it once with typescript and twice with python, I learned a lot and I am looking forward to your new article.

1 ^ | v • Reply • Share ›

**Saad** • 3 months ago

I had another solution to this, I tried this code and it actually gave me the same result !

```
ar = ActivationRecord(  
    node.name,  
    "<procedure>",  
    self.call_stack.top.nesting_level + 1,  
)
```

My code is so different I know, and I hope to understand the different.

^ | v • Reply • Share ›

**Caleyo** • 4 months ago • edited

19 lessons covering stuff like functions and procedures but still no word on conditions. I think they should have been covered right after the abstract syntax tree.

^ | v • Reply • Share ›

**Lucas Dutra** • 5 months ago • edited

Nice your tutorial, very big also.

But I realize you didn't finish it yet, so, passed a year and no more parts. Finished here in 19.

I think it would be better, for now, to read a book about compilers instead of waiting for the completeness of this tutorial.

I expect you could help a lot of people and expect you can finish

^ | v • Reply • Share ›

**Nikola Stojiljkovic** • 6 months ago

Thanks a lot for the articles. I've found them very useful and inspiring.

I followed the articles while I was working on my own implementation. I implemented everything before looking at any solutions. As a result there are some differences in my approach, but it's similar overall.

I've done this in C++. My code is available here: <https://github.com/freezing...>

Disclaimer: I took a lot of shortcuts given that my focus was on understanding how interpreters work, rather than code quality.

I'm most-likely going to stop working on this project for the time being, but I may pick it up again in the future.

^ | v • Reply • Share ›

**Alex Emelyanov** • 8 months ago • edited

This is a really great series a learned a lot through this. And this allows me to implement a simple interpreter and integrate it in a project on my work <https://github.com/augerai/...> (this is a second interpreter version). Initial version was created when I didn't meet this awesome series and was tricky and very complicated and hard to extend and understand.

^ | v • Reply • Share ›



Jay • a year ago • edited

Thank you very much for putting your time to write this. It is very helpful. It takes me one and a half month to go through the existing blogs. Looking forward to the next post.

I am in Toronto as well. Stay healthy, stay safe.

I really like small stories or quotes you have at the beginning of some posts. Some of the quotes are Chinese proverb, I enjoy translating them back to Chinese.

^ | v • Reply • Share ›



Tilak Madichetti • a year ago

this blog helps me a lot

^ | v • Reply • Share ›



solstice333 • a year ago

@**rspivak** thanks for putting in the effort and time to writing this. Looking forward to the next one in the series!

^ | v • Reply • Share ›



lee qiaoping • a year ago

I've read all previous articles and I am very expecting on the flow control parts, like if/else, loop etc...

Thanks for the hard work!

^ | v • Reply • Share ›



Alan Telles • a year ago

Hello, Ruslan! First, let me tell you that your tutorial helped me a lot to redesign my language. I'm implementing the tutorial in Pascal and my lang is like Python. It's been a challenge but a cool challenge. I'm doing my research! Waiting for the next part! Thank you!

^ | v • Reply • Share ›



Irwin Rodriguez • a year ago

Hi Ruslan, thank you for this great series. Do you have any printed version?

^ | v • Reply • Share ›



rspivak Mod → **Irwin Rodriguez** • a year ago

Hi Irwin,

You're welcome :)

I don't have any printed version yet, but I plan to create one after I'm done with the series.

^ | v • Reply • Share ›



Irwin Rodriguez → **rspivak** • a year ago

Awesome! Either printed or PDF would be great! I'll be on touch by subscribing into your mail list!



^ | v • Reply • Share ›



Πιστός Στις Αξίες • a year ago

Thanks for sharing. I want to use the code for making something for my school. Greetings!

^ | v • Reply • Share ›



Junan Lu • a year ago

Thank you for your sharing, I've learned a lot from your awesome blog. Looking forward to your next article...

^ | v • Reply • Share ›



Aleksandr • a year ago

A week ago, I came across this series of articles looking for information on how to implement a simple tokenizer that I thought I will expand easily into my tiny text processing language. Damn it, I was naive! I'm a self-taught programmer, and I've never done anything like this before, so it was an intense week! I think I've learned enough basics to continue my

I've never done anything like this before, so it was an intense week! I think I've learned enough basics to continue my studies on my own. But I hope to see a sequel to this series soon! Thank you Ruslan!

^ | v • Reply • Share ›



Maddy • a year ago

I love your content so far ! Thanks a ton for value you provided :) I want to ask you if I can expect decision making, looping and recursion will be taught as they are sort of the basic in every programming language

^ | v • Reply • Share ›



selim öztürk • 2 years ago

thank you Ruslan

^ | v • Reply • Share ›

✉ Subscribe Add Disqus to your siteAdd DisqusAdd Do Not Sell My Data

🏠 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。