自己动手写Python虚拟机

Write Your Own Python Virtual Machine

海 纳 编著

北京航空航天大学出版社
BEIHANG UNIVERSITY PRESS

自己动手写 Python 虚拟机

海　纳　编著

北京航空航天大学出版社

内 容 简 介

本书按内容分为六个部分，第一部分介绍语言虚拟机的基本概念，并实现字节码解释器；第二部分，实现内嵌类型，如整数、字符串、列表和字典等；第三部分，实现了函数；第四部分，实现自定义类、对象和方法；第五部分，实现垃圾回收，也就是自动内存管理；第六部分，模块和迭代。本书的章节内容之间都有很强的依赖性，后面的章节内容都是在前面章节的基础上去实现的，所以读者必须按部就班地从前向后阅读，才能保证阅读的流畅。

本书适合的人群包括：在校大学生（可以通过本书掌握很多计算机工作运行的核心知识），以及对编译器，编程语言感兴趣的人。

图书在版编目(CIP)数据

自己动手写 Python 虚拟机 / 海纳编著. -- 北京 : 北京航空航天大学出版社，2019.6

ISBN 978-7-5124-2975-8

Ⅰ. ①自… Ⅱ. ①海… Ⅲ. ①软件工具—程序设计 Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字(2019)第 055841 号

自己动手写 Python 虚拟机

海 纳 编著

责任编辑 剧艳婕

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(邮编 100191) http://www.buaapress.com.cn

发行部电话：(010)82317024 传真：(010)82328026

读者信箱：emsbook@buaacm.com.cn 邮购电话：(010)82316936

三河市华骏印务包装有限公司印装 各地书店经销

*

开本：710×1 000 1/16 印张：21.25 字数：453 千字

2019 年 6 月第 1 版 2019 年 6 月第 1 次印刷 印数：2 000 册

ISBN 978-7-5124-2975-8 定价：79.00 元

前　言

编程语言是每个程序员每天都要使用的基本工具，现代的主流编程语言以 Java、javascript 和 Python 为代表，都是运行在语言虚拟机之上的，很多人都很想知道语言虚拟机的内部构造。我从 2017 年开始在知乎撰写专栏《进击的 Java 新人》，专栏中对 Java 语言虚拟机的字节码和垃圾回收做了一些简单的介绍，很多读者发私信给我，表示非常想知道更多的细节。在这样的背景下，我开始了本书的写作。我希望在这本书中和读者一起从零开始构建一个完整的编程语言虚拟机，它将会涉及到字节码的解析执行、对象系统、语言内置功能和垃圾回收等多个主题。

本书适合的人群包括：

1. 在校大学生，大一大二的同学可以通过本书掌握很多计算机工作运行的核心知识 。

2. 对编译器和编程语言感兴趣的人。相比起直接将一门语言编译成机器码，将其编译为虚拟机上的字节码文件会简单很多，所以掌握一门虚拟机字节码，甚至自己实现一个虚拟机对学习编译器、了解编程语言特性有很大的帮助。

本书的内容虽然很新颖，但是对读者门槛的要求并不高。读者只要简单地掌握一些 Python 或者某一门类 C 语言（例如 Java）即可。在本书中，我选择了使用 C++ 来实现语言虚拟机。这主要是由于在内存操作方面，C++可以更精准地表达作者的意图。C++是一门很难的语言，相比起 Java、Python 和 PHP 等语言，流行度也不高，但是读者不必有畏难情绪，本书在使用 C++ 的时候是比较克制的。本书并没有使用很多 C++ 的高级技巧，最多只涉及到类和一点点的模板编程的知识。C++是一门多范式的编程语言，我们不可能在一个工程中使用所有的编程范式。本书中所涉及的代码，读者只需要有任何一门面向对象的语言的编程经验即可顺利阅读。

如何使用本书：

本书共分为六个部分，第一部分介绍语言虚拟机的基本概念，并实现字节码解释器；第二部分，实现了内嵌类型，如整数、字符串、列表和字典等；第三部分，实现了函数；第四部分，实现自定义类、对象和方法；第五部分，实现垃圾回收，也就是自动内存

管理；第六部分，模块和迭代。其中第二、第三和第四部分的实现并不是完全独立的，而是相互嵌套依赖的。比如完整的对象系统必然依赖函数，而 Python 中的函数本身也是对象，这就产生了循环依赖，解决这个问题的办法是先实现一套相对简单的对象系统，然后基于此也实现一套简单的函数系统，再回过头来补充完善对象系统……这样螺旋式地上升，最终完成整个系统的搭建。

本书章节的内容之间都有很强的依赖，后面章节的内容都是在前面的章节的基础上去实现的。所以读者必须按部就班地从前向后阅读，才能保证阅读的流畅。本书为了节约篇幅，对于一些逻辑比较简单的代码，就都省略了。读者可以在 https://gitee.com/hinus/pythonvm 里找到全部的代码，包括该项目最近的更新以及各种提交记录。在提交记录中，读者可以清晰地看到本项目的进化过程。

感谢出版社编辑剧艳婕的耐心审校，尤其还要感谢专栏《进击的 Java 新人》的读者，是你们的精彩评论和学习反馈引发了这本书的创作。

实现一个高效的编程语言虚拟机是一个十分复杂的问题，从 Hotspot 虚拟机的发展过程中就可以看出来。书中难免有讹错纰漏之处，欢迎读者及时指出。书中如果有描述不清的地方，也欢迎读者来信交流，可发至邮箱：hinus@163.com。

作　者

2019 年 4 月

目 录

第1章

编程语言虚拟机

本章简单地介绍一下编程语言的发展历史，什么是编程语言虚拟机，以及为什么要引入虚拟机的概念。

1.1 编程语言的发展

计算机技术发展至今，已经有很多编程语言了，这些语言的工作原理各不相同。根据它们与硬件平台的远近关系，可以把编程语言粗略地划分为以汇编语言为代表的底层语言和以C++、Java为代表的高级语言。

汇编语言的特点：直接与硬件平台提供的寄存器、内存和IO端口打交道，功能十分强大。早期操作系统镜像的加载和初始化经常使用汇编来实现；语言助记符（例如mov、add）几乎与CPU指令一一对应，使用汇编语言几乎可以不考虑编译器的影响，这就让编程人员对代码有绝对的控制权。但是汇编语言表达能力不强，开发效率低。

为了提高应用程序的开发效率，人们发明了高级语言。C语言是非常重要的一种语言，保留了内嵌汇编，并且可以通过链接器将汇编语言开发的模块与C语言开发的模块链接在一起；同时C语言的指针也保留了汇编语言中内存操作的逻辑。它是一门承上启下的语言，大多数操作系统都是用C语言开发的，说C语言是计算机行业的基石也不过分。

随着应用软件的规模越来越大，面向对象的编程思想开始流行，面向对象的语言也应运而生，典型的代表就是C++。面向对象的编程方式可以让开发者以模块化、对象化的思想进行设计和开发，大大提高了编程语言的抽象能力。到目前为止，面向对象的编程方式仍然是当今世界的主流编程方法。

然而使用C++的时候，编程人员要十分小心地使用内存，因为稍不注意就容易引起内存泄漏，例如以下代码：

```
1    void foo() {
2        Data * data = new Data();
3        //...
```

```
4         //many other codes
5         if (some_condition())
6             return;
7
8         //...
9         //many other codes
10        delete data;
11        return;
12    }
```

在 some_condition 条件满足的情况下，函数 foo 方法就直接返回了，漏掉了 delete 语句。这一块内存没有被释放，但却没有任何变量引用它。也就是说，应用程序再也无法正常访问这块内存了，这就是内存泄漏。如果函数体比较小，逻辑相对简单，程序员一般不会犯这种错误，但如果逻辑比较复杂，尤其是多人多版本维护同一份代码的情况下，自己添加的 return 逻辑将别人添加的 delete 跳过去的情况就非常常见。

还有一个痛点是跨平台。当前主流的体系结构包括 x86 和 arm，操作系统包括 Windows，Linux，Android 等，跨平台是指同一份代码可以在多种不同的体系结构和操作系统上正确执行。C/C++这类语言是静态编译的，它们编译完成后就是直接可执行的程序（例如 Windows 系统上的 exe 文件），可执行程序中的代码段里的内容是与平台直接相关的，在 x86 系统上，就会产生 x86 的机器指令，在 arm 平台上，就会产生 arm 的机器指令。另外，还要考虑编译器和操作系统以及运行时库的影响。同样的 C++代码，不同版本的编译器和操作系统会产生不同的代码，同时应用程序所依赖的动态链接库也会有不兼容的情况。静态编译为应用程序的分发和部署带来了困难。

为解决这些问题，编程语言虚拟机应运而生。

1.2 编程语言虚拟机

1. 屏蔽硬件差异

编程语言虚拟机的一个重要功能就是屏蔽硬件差异。以 Java 为例，Java 源代码文件会被 Javac 先编译生成 class 文件，多个 class 文件可以集中在一起，生成一个 jar 文件。通常，一个模块会压缩成一个 jar 文件，而应用程序就以 jar 包的方式分发和部署。

class 文件的格式是固定的，它的代码段里全部是 Java 字节码。不管在什么硬件环境下，相同的字节码得到的执行结果一定是相同的。字节码的设计类似于 CPU 指令，它有自己定义的数值计算、位操作、比较操作和跳转操作等。因此，这种专门为某一类编程语言所开发的字节码及其解释器被合并称为编程语言虚拟机。

可以借助一个例子来了解 Java 虚拟机的工作原理，如以下 Java 代码：

```
1    void foo() {
2        int a = 2;
3        int b = 3;
4        int c = a + b;
5    }
```

它编译的 Java 字节码如下所示：

```
1    0: iconst_2
2    1: istore_1
3    2: iconst_3
4    3: istore_2
5    4: iload_1
6    5: iload_2
7    6: iadd
8    7: istore_3
9    8: return
```

这个字节码的的执行过程如图 1.1 所示。Java 的执行过程可以通过高级数据结构的栈来实现，而不必关心 CPU 的具体体系结构。

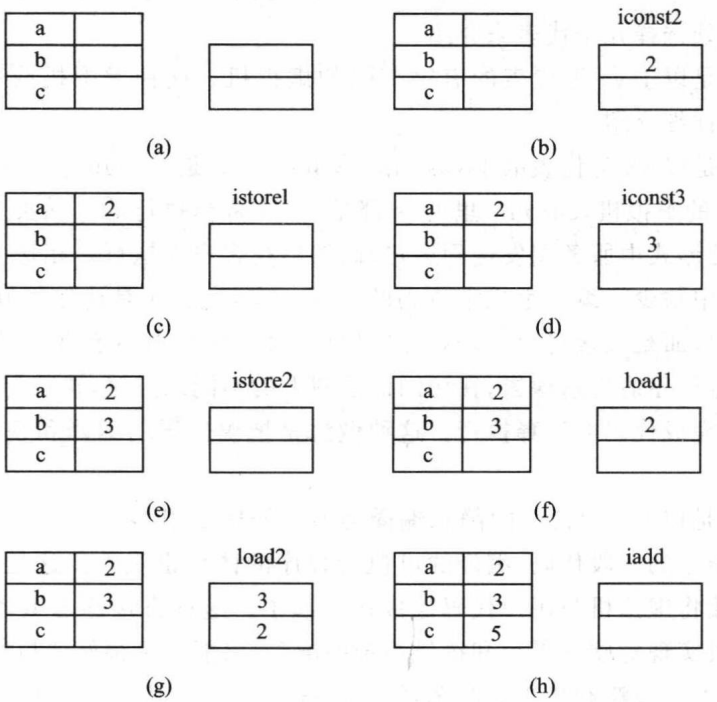

图 1.1　JVM 执行过程示意图

对字节码文件进行加载、分析，执行的逻辑都封装在 Java 语言虚拟机里。在不同的硬件平台和不同的操作系统上，Java 语言虚拟机的实现各不相同，但是，它提供的字节码执行器的功能是完全相同的。Java 早期推广的时候，曾经以“write once，run everywhere”作为重要的特性进行宣传，主要就是指 Java 语言虚拟机屏蔽硬件差异的能力。

2. 自动内存管理

使用 C++进行开发时，开发人员要十分注意内存的使用，避免不合理的内存分配和内存泄漏问题。如果能够自动对不被引用的内存进行回收和整理，就可以彻底避免内存泄漏，这就是垃圾回收机制。

编程语言自动内存管理的研究开始得比较早，在当今主流的带有垃圾回收的编程语言（例如 Java、Python、ruby 和 Go）被开发出来之前，垃圾回收的算法就已经非常成熟了。

大体上，垃圾回收可以分为引用计数和 Tracing GC 两大类，其中引用计数的代表就是 CPython，也就是平常最常使用的社区版 Python。而大多数编程语言虚拟机都使用 Tracing GC。Tracing GC 的家族非常庞大，这里不再展开讨论。在本书的自动内存管理一章，将会深入地讨论这个问题。

3. 编译和执行

在屏蔽硬件差异小节，只举了 Java 字节码的例子。其实，编程语言还可以有更多的实现，此处选择几个代表来介绍。

第一类，是以 Java 为代表的有字节码的虚拟机。这种虚拟机前边已经介绍过了，这里不再详细介绍。

第二类，是以 v8 为代表的 Javascript 虚拟机。v8 是 Google 公司开发的，用于执行 Javascript 的虚拟机，chrome 里的 js 都是由 v8 解释执行的。网页上的 js 代码都是以源代码的形式由服务端发送到客户端，然后在客户端执行。相比 Java 的执行过程，这一过程中缺少了编译生成字节码的步骤。实际上 v8 是比较特殊的，根本不需要生成字节码，而是直接将源代码翻译成树形结构，称其为抽象语法树。然后，v8 的执行器就通过后序遍历这棵树，在访问语法树上的不同结点时，执行与该结点相对应的动作，最终完成代码的解释执行。这种做法是把源代码的编译和程序的执行直接绑定在一起。

第三类，是以 Go 为代表的静态编译类型。如果使用 Go 语言进行编译的话，会发现即使是很小的一段代码，编译的可执行程序的体积也很大。这是因为，Go 在编译的时候直接将虚拟机与用户代码链接在一起了。这种直接静态编译的好处是，既能通过虚拟机实现对硬件平台和操作系统的屏蔽，又能提供很好的执行效率。

可见，在编译策略和执行器的选择上，编程语言的实现是有多种选择的。没有哪一种选择是十全十美的，开发者只能根据编程语言所面临的场景以及所要解决的问

题来决定。

4. Python 的策略

Python 比较灵活，一方面它规定了自己的字节码，但又不要求程序必须以字节码文件(pyc)来发布；它完全支持源代码的直接执行。

本质上，在 Python 虚拟机内部，源代码也是先编译成字节码然后再执行的，也就是说，Python 的编译器是 Python 虚拟机的一部分。它不像 Java 虚拟机，Javac 用于编译，和执行是相分离的。可以回忆一下 Python 中的 eval 功能，eval 就是调用了 Python 内置的编译器来对字符串进行编译的。

另外，Jython 是一种 Java 实现的 Python 语言，它的原理与 CPython 大不相同。它放弃了 Python 的原生字节码，直接将 py 源代码文件翻译成了由 Java 字节码组成的 class 文件。而 class 文件是可以直接在 Java 虚拟机上执行的，这样一来，Python 代码就可以自由地使用各种强大的 Java 类库。通过编译，Jython 实现了 Python 与 Java 的无缝衔接。

1.3 开发环境

可见，仅仅依赖编译技术，也可以很好地实现执行 Python 代码的能力；所以，编译技术在 Python 虚拟机中占据非常重要的地位。但是，考虑到编译器的开发，学习曲线陡峭，许多人在没有体会到乐趣之前，就淹没在各种抽象的概念里了，所以本书先不关注编译的过程(只在第 2 章简单地介绍一下词法分析、文法分析以及抽象语法树等基本概念)，而只将注意力集中在实现一个字节码文件的执行器。在这个过程中，采取小步快走的方式，让读者可以快速地看到成果，可以看到每一点改进，这样的方式才有利于坚持学习下去。

本书选择使用 C++实现 Python 虚拟机，不使用 C 或者 Java。主要是考虑到，C++比 C 的表达能力更强，开发速度更快；相比 Java，C++直接操作内存的能力更强，对整个程序的内存使用更好、更精细，这是开发一个内存管理系统的必要条件。

我所用的平台是 x86 64 位的 ubuntu，读者完全可以使用其他体系结构的其他操作系统来进行实验。因为代码使用了 cmake 来进行管理，即使读者使用 Windows 系统或者 MacOS 都不会有太大的问题。

第2章

编译流程

本章先通过一个小的类C语言来讲解编译的流程。这个编译的流程与Python中的编译流程是完全一样的，此处通过这个例子了解编译的过程。

由于本书以实战为主，因此在这里不打算讲解各种严肃的名词，尽量使用最直观的方式来描述每一步要做的事情。

回忆我们写的程序，不管是Java文件，还是Python源代码文件，本质上都是一种文本文件。编译的过程就是把文本文件翻译成一个可以被Java虚拟机或者Python虚拟机打开、识别和执行的文件。例如，Java文件通过Javac可以翻译为class文件，而py文件经过提前编译，也会生成pyc文件。class文件是由Java字节码组成的，pyc文件是由Python字节码组成的，所以也称这两种文件为字节码文件。

2.1 Python字节码

在开始研究编译流程之前，先大概了解一下Python字节码的设计。与Java字节码非常相似，Python字节码的执行也是基于栈的。对于第一章Java的例子，此处可以验证一下Python是什么样的。

在命令行运行Python，然后就可以交互式地执行Python代码了。执行以下代码，可以看到如下结果：

```
1    >>> def foo():
2    ...     a = 2
3    ...     b = 3
4    ...     c = a + b
5    ...     return c
6    ...
7    >>> import dis
8    >>> dis.dis(foo)
9              2           0 LOAD_CONST               1 (2)
10                         3 STORE_FAST               0 (a)
11
```

```
12      3         6 LOAD_CONST               2 (3)
13                9 STORE_FAST               1 (b)
14
15      4        12 LOAD_FAST                0 (a)
16               15 LOAD_FAST                1 (b)
17               18 BINARY_ADD
18               19 STORE_FAST               2 (c)
19
20      5        22 LOAD_FAST                2 (c)
21               25 RETURN_VALUE
```

对比 1.2 节的 Java 虚拟机的例子，可以看到 Python 的字节码与 Java 的字节码非常相似。执行的过程也如图 1.1 所示的那样，使用一个变量表和一个操作数栈来完成所有字节码的执行。第 1 章中已详细介绍过这个执行过程，这里不再重复。

dis 模块的功能是反编译 Python 字节码。在上面的例子中，通过 dis 反编译了 foo 这个函数。Python 字节码有两种类型，一种带参数，一种不带参数。在真实的内存中，每个字节码都有一个编号，这个编号叫做操作码(Operation Code)，只占 1 个字节。不带参数的字节码只有操作码，所以它的大小就是 1 个字节；带参数的字节码，最多也只能带一个参数，而每个参数占 2 个字节，所以带参数的字节码就占 3 个字节。例子中的 LOAD_CONST 和 STORE_FAST 就是带参数的字节码，而 BINARY_ADD 则是不带参数的字节码。

粗略地了解了 Python 字节码以后，再来讨论 py 文件是如何被翻译成这些字节码的。

2.2 词法分析

如何把文本文件翻译成字节码文件呢？第一个步骤就是从文本文件中逐个字符地去读取内容，然后把字符识别成变量、数字、字符串、操作符和关键字等。这些变量、字符串和数字是组成程序的基本元素，它有一个专用的名字，叫 token。

把文本文件中的一串字符识别成一串 token，就是我们要解决的第一个问题。先看一个例子，创建一个文本文件，例如叫 test_token.txt，其中只包含一行表达式：

```
12 * 48 + 59
```

这个表达式是由 5 个 token 组成的，分别是数字 12、乘号(*)、数字 48、加号(+)和数字 59。针对这个文本文件，可以写一个程序，不断地读入字符，并把其中的 token 识别出来。代码如下：

```c
#include <stdio.h>

#define INIT 0
#define NUM 1

int main() {
    FILE * fp = fopen("test_token.txt", "r");
    char ch;
    int state, num = 0;

    while ((ch = getc(fp)) != EOF) {
        if (ch == ' ' || ch == '\n') {
            if (state == NUM) {
                printf("token NUM : %d\n", num);
                state = INIT;
                num = 0;
            }
        }

        else if (ch >= '0' && ch <= '9') {
            state = NUM;
            num = num * 10 + ch - '0';
        }

        else if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            if (state == NUM) {
                printf("token NUM : %d\n", num);
                state = INIT;
                num = 0;
            }

            printf("token operator : %c\n", ch);
            state = INIT;
        }

    }

    fclose(fp);
    return 0;
}
```

在上面的程序中，遇到加、减、乘、除操作符时，就可以直接输出这个操作符。唯