互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

第1天 来，我们一起做些什么吧

——某大学研究室内

C 话说，我现在正在写一本新书。
H 老师，您这次写的是什么主题的书呢？
C 是一本和编译相关的书。确切地说，是关于语言处理器的书。
F 这样啊，这次是要写成一本教科书吗？
C 不，出版社要求我这次写得通俗些，所以这本书的内容会比教材来得简单。
H 那这次还会像前一本书[①]那样，通过对话形式进行解说吗？
C 这个问题现在还没有确定。有人赞成用对话的形式，但也有人反对。
F 老师，那这次的新书中会出现哪些人物呢？肯定会有 H 吧，毕竟这里他最年长。
H 哎呀，别这么说，就算没有我也没关系。
F H 肯定会出现啦。至于还会有哪些人，真是很期待呀。此外，M[②]那样称职的角色也必不可少。这次选谁才好呢？

设计程序时使用的语言称为程序设计语言。如 Java 语言、C 语言、Ruby 语言、C++ 语言、Python 语言等，都是程序设计语言。

程序员必须使用与各程序设计语言相匹配的软件来执行由该语言写成的程序。这种软件通常称为语言处理器。本章将首先说明语言处理器的基本概念。

1.1 机器语言与汇编语言

——不久后

A 该不会是要让我来扮演 M 的角色吧？真是这样倒也没问题，M 一直也很关照我。
C 不，所有出现的人物都是虚构的，不必在意。

有些程序设计语言无需借助软件执行，也就是说，它们不需要语言处理器。这些语言称为机器语言。机器语言可以由硬件直接解释执行，理论上不必使用软件。

① 千叶滋《面向方面程序设计入门》技术评论社，2005 年。
② 这里指的是在注 1 提到的书中出现的角色 M。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

然而，机器语言书写的程序只有载入内存后才能通过硬件执行。因此用户在实际使用时，必须先通过软件从磁盘文件中读取机器语言程序，再将它复制至内存。不过，这类程序称不上是语言处理器，通常称为操作系统（Operating System，OS）。

A 我先打个岔，如果说操作系统是用于复制的软件，机器语言就该是其中的程序了吧。

F 你是想问，机器语言是不是需要通过某种软件来复制到内存吧?

C 当然需要了。这叫做引导装载程序。

A 老师，我想知道的是这个引导装载程序是怎样被复制到内存中的呢?

F 小 A，引导装载程序会事先写在内存中，无需复制。计算机在启动时会首先执行这个程序。

C 没错，即使切断电源，引导装载程序依然会留在内存中。

A 那为什么不一开始就把操作系统写入内存呢?

S 那样的话，升级操作系统将会变得很麻烦。

F 而且也无法实现 Windows 和 Linux 双操作系统启动。

C 嗯。不过要是断电后数据也不会丢失的高速内存能得到普及，预先将操作系统写入内存的计算机系统也会出现吧。

汇编语言与机器语言是很容易混淆的概念，但两者并不相同。机器语言写成的程序本质上是一个位数很长的二进制数字。由于它不易于阅读，人们常通过汇编语言程序来表述这个巨大的数字，使其更易于理解。因此，如果要执行汇编语言写成的程序，用户通常需要使用软件将其转换为机器语言。这种软件称为汇编程序（assembler）。汇编程序可以说是一种最基本的语言处理器。

1.2 解释器与编译器

语言处理器可大致分为解释器与编译器两种。这两类语言处理器的执行原理有很大差异。

● 解释器

解释器根据程序中的算法执行运算。简单来讲，它是一种用于执行程序的软件。如果执行的程序由虚拟机器语言或类似于机器语言的程序设计语言写成，这种软件也能称为虚拟机。

● 编译器

编译器能将某种语言写成的程序转换为另一种语言的程序。通常它会将原程序转换为机器语言程序。编译器转换程序的行为称为编译，转换前的程序称为源代码或源程序。如果编译器没有把源代码直接转换为机器语言，一般称为源代码转换器或源码转换器（source code translator）。

程序设计语言提供了何种类型的语言处理器不一而论，一些具有解释器，另一些则会提供编译器。例如，尽管 C 语言也提供了解释器，但却很少使用。C 语言通常直接通过编译器转换为机器语言执行。转换后得到的机器语言程序会暂时保存至某个文件，需要借助操作系统来执行。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

另一方面，Common Lisp 或 Haskell 等语言一般会同时提供解释器与编译器，供用户根据需要选用。

有些语言混用解释器与编译器。通常，Java 语言首先会通过编译器把源代码转换为 Java 二进制代码，并将这种虚拟的机器语言保存在文件中。之后，Java 虚拟机的解释器将执行这段代码。

传统的狭义的编译器将会以文件形式保存转换后的程序。因此，只要源程序没有变更，编译就仅需执行一次，执行时间也会缩短。然而，一些编译器并不保存转换后的程序文件。这种编译器常见于解释器内部。

大多数 Java 虚拟机为了提高性能，会在执行过程中通过编译器将一部分 Java 二进制代码直接转换为机器语言使用。执行过程中进行的机器语言转换称为动态编译或 JIT 编译（Just-In-Time compile）。转换后得到的机器语言程序将被载入内存，由硬件执行，无需使用解释器。

编译器的用途多样。如上所述，它能够直接在解释器内部执行。此外，编译器的作用也不局限于将源程序转换为机器语言。例如，Ruby 语言的解释器内部会通过编译器来执行预处理工作，将源程序转换为类似于 Java 二进制代码的虚拟机器语言程序。解释器真正执行的是这种经过编译的语言。这种设计提高了执行性能。

C 最近在解释器内部编译的例子越来越多，解释器的定义也变得模糊了呢。

F 是呀。不过前面提到的 Java 源代码将首先经过编译这一点，恐怕很多人并不知道吧。

H 的确如此，如果使用 Eclipse 开发 Java 程序，开发者很难看到编译过程。

F Eclipse 其实已经把编译器与编辑器整合了，编译器会在开发者书写代码的同时执行编译，就好像编译始终能即时完成。

C 对，开发者要意识到代码已被编译反而是一件难事。

过去人们提到编译器时，首先会联想到费时的编译过程。不过由于编译后实际执行的是机器语言，因此执行速度很快。而对于解释器，人们通常认为它会在程序输入的同时立即执行，执行速度较慢。这就是两者的基本区别。现代的解释器内部常采用各种类型的编译器，已经越来越没有必要将解释器与编译器区分看待。

C 另外，编译器是否将源代码转换成了机器语言，并不那么易于分辨呢。

A 只要编译后的文件双击后能够运行，它就是机器语言了。

S 咦，但是 Java 编译后的 `.jar` 文件大都能双击运行不是吗？

H 嗯，如果我说 `.jar` 文件的内部其实是机器语言，大概也会有人相信吧。

C 当然了，`.jar` 文件内保存的是 Java 二进制代码。操作系统将会在后台启动 Java 虚拟机，并通过它来运行 `.jar` 文件。

F Android 系统也是这种机制，它采用了名为 Dalvik 的虚拟机。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

1.3 开发语言处理器

本书将为极为简单的脚本语言开发语言处理器。由于对象是脚本语言，所以如果按上一节的分类方式，本书开发的语言处理器属于解释器。不过，该解释器内部将采用编译器来提高性能，因此本书也将涉及开发编译器的一些基本知识。本书不包含代码优化之类的技巧，因此不会介绍诸如编译器在将程序转换为机器语言时，如何提高机器语言的执行效率等内容。

F 脚本语言这个词的含义有些模糊不是吗？
C 嗯，这的确是一个无法回避的问题。
H 要回答脚本语言是怎样的程序设计语言，实在是不容易。
C 总之，我们并不是要设计 C 语言那样的语言。不过，这类主题的书常会选择 C 语言的某些简化版本作为研究对象呢。
F 本书会包含通过正则表达式实现模式匹配的语法功能吗？
C 我不打算介绍这些。
F 本书中出现的语言，会像 Perl 那样，同一种逻辑可以通过多种方式表达吗？
H 熟悉之后，只需数行就能写出复杂的功能，这也是脚本语言的一个特点了。
C 你们当然可以增加语法的种类，不过这就留作课后作业吧。毕竟不同语法的本质是相同的。
H 也就是说不会介绍这部分内容了吗？
C 是的。这只会平白增加篇幅而已。
F 那本书使用的语言还能称为脚本语言吗？
C 想问的是这个啊？这种语言支持动态数据类型，无需事先声明变量，且通过解释器运行。其实本书的主题应该是以现代的手法来设计现代语言。
A 这样一来，这本书会变成什么样子呢？或许会有人说书的标题与内容不符了吧。

本书将设计的语言命名为 Stone 语言。实现该语言的开发语言是 Java 语言。因此，Stone 语言也是一种运行于 Java 虚拟机的语言。

H 老师，还是说明一下“实现”这个词的含义比较好吧？
C 这里的实现（implementation）指的是通过程序来实现某种功能。把它理解为书写程序也可以。
F 说起来，Stone 语言的命名灵感是来自 Perl 语言和 Ruby 语言对吧？
C 没错。它称不上是宝石，顶多算是小石子，因此取名为 Stone。

Stone 语言运行于 Java 虚拟机，并不轻巧。之所以选择 Java 语言，是为了以面向对象的方式设计语言处理器。语言处理器的复杂度适中，常用于实验或论证各种语言范型的性能。

例如，Haskell 语言或 OCaml 语言之类的函数型语言，非常适合开发语言处理器。面向对象语言也是如此。本书在讲解时，默认读者十分了解面向对象语言，尤其是 Java 语言的基本编程方式。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

A 如果是要使用面向对象语言，Ruby 语言或 Scala 语言这些不可以吗？

C 这个嘛……它们可能会赶跑一部分读者，编辑或许会否决这个提议的吧。

H 那用 C 语言和 yacc 来实现的话如何？老师您觉得这样可以吗？

C 嗯，C 语言本身没什么不好，但要实现稍微复杂些的语言处理器时，就不得不使用各种不同的编程技术。最终写出的 C 语言程序会具有面向对象风格，那还不如从最开始就使用面向对象语言。

S 我倒是觉得以函数式语言风格来写 C 语言代码也挺好。

H 如果要写一本设计 Tiny C 编译器的书，C 语言会是个不错的选择吧。

C 此外，这里不会使用 yacc 相关的外部工具。我打算用其他方法来设计。

1.4 语言处理器的结构与本书的框架

无论是解释器还是编译器，语言处理器前半部分的程序结构都大同小异。如图 1.1 所示，源代码首先将进行词法分析，由一长串字符串细分为多个更小的字符串单元。分割后的字符串称为单词。之后处理器将执行语法分析处理，把单词的排列转换为抽象语法树。至此为止，解释器与编译器的处理方式相同。之后，编译器将会把抽象语法树转换为其他语言，而解释器将会一边分析抽象语法树一边执行运算。

F 首先需要把源代码转换为抽象语法树没错吧？

C 程序的分析结果能由抽象语法树表现，因此无论是解释器还是编译器都需要用到抽象语法树。

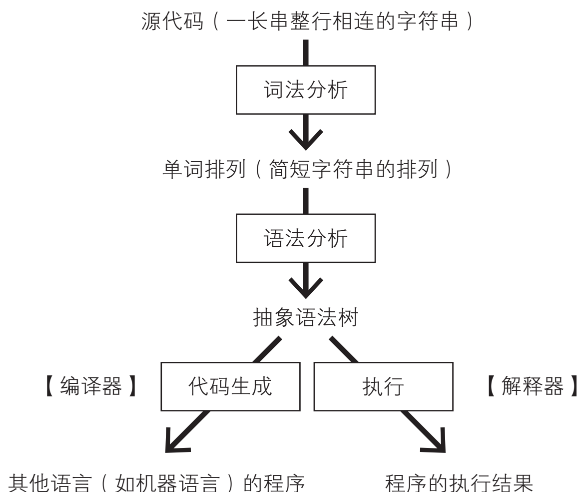

图 1.1　语言处理器内部的处理流程

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

今后，本书将根据这一流程开发 Stone 语言的处理器。各章内容如下所示。

第 1 部分　基础篇

设计 Stone 语言的解释器。第 2 ~ 8 章将实现一个具有基本功能的解释器。第 9 ~ 10 章将介绍一些高级内容。

- **第 1 章（第 1 天）**

  本章。

- **第 2 章（第 2 天）**

  第 2 章将设计 Stone 语言，决定 Stone 语言需要具备哪些语法功能。

- **第 3 章（第 3 天）**

  第 3 章将设计词法分析器，介绍通过正则表达式实现词法分析的方法。

- **第 4 章（第 4 天）**

  第 4 章将讲解抽象语法树，并通过 BNF 表达 Stone 语言的语法。

- **第 5 章（第 5 天）**

  第 5 章将利用非常简单的解析器组合子库来创建语法解释器。解析器组合子库的内部结构将在第 17 章进行说明。

- **第 6 章（第 6 天）**

  第 6 章将设计一种极为基本的解释器。在这一章结束后，解释器将能够实际执行 Stone 语言写成的程序。本书采用了 GluonJ 这一系统来设计解释器的程序，因此这一章还会简单介绍 GluonJ 的使用方法。

- **第 7 章（第 7 天）**

  第 7 章将增强解释器的功能，使它能够执行程序中的函数，并且支持闭包语法。

- **第 8 章（第 8 天）**

  第 8 章将为解释器增加 static 方法的调用支持，使 Stone 语言能像 Java 语言那样调用静态方法。

- **第 9 章（第 9 天）**

  第 9 章将为 Stone 语言新增类与对象的语法。本章将使用闭包来实现该功能。

- **第 10 章（第 10 天）**

  第 10 章将为 Stone 语言增加数组功能。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

第 2 部分　性能优化篇

第 2 部分将对第 1 部分设计的 Stone 语言解释器进行性能优化。其中，第 13 章将介绍如何设计 Stone 语言的编译器，帮助提高性能。如果读者仅对编译器的设计方法感兴趣，只需阅读第 11 章与第 13 章即可。

● **第 11 章（第 11 天）**

程序不应在访问变量时每次都搜索变量名，而应首先搜索事先分配好的编号，提高访问性能。

● **第 12 章（第 12 天）**

同样地，程序在调用对象的方法或引用其中的字段时，也不应直接搜索其名称，而应搜索编号。此外，第 12 章还会为 Stone 语言的解释器增加内联缓存，进一步优化性能。

● **第 13 章（第 13 天）**

Stone 语言的解释器也采用了中间代码解释（或虚拟机）的机制。Stone 语言写成的程序将首先被转换为中间代码（或二进制代码），解释器执行的其实是转换后的中间代码。Ruby 等语言也采用了这样的方式。第 13 章还将介绍如果要设计一个能把 Stone 语言转换为机器语言的编译器，需要做哪些准备。

● **第 14 章（第 14 天）**

最后，为了提高性能，Stone 语言有必要支持静态数据类型，并根据数据类型的不同进一步优化性能。在执行具有静态数据类型的 Stone 语言程序时，编译器可以先将其转换为 Java 二进制代码，再直接由 Java 虚拟机执行该程序。第 14 章还会为编译器增加类型检查功能，在执行程序前检查是否存在类型错误，并同时提供类型预判功能。这样一来，即使程序没有显式地声明数据类型，Stone 语言的解释器也能推测并指定合适的类型。Scala 等一些语言也采用了这一机制。

第 3 部分　解说篇（自习时间）

第 3 部分将介绍一些在开发 Stone 语言过程中没能涉及的进阶主题。第 15、16 章的内容是大多语言处理器相关教材中都会讲解的基础知识。

A 咦，前 14 章就把书名所讲的内容都介绍完了呢。

C 嗯，的确如此。

A 这么做是为了博人眼球吗？

H 小 A，不能这么挑刺哦。

F 不过其实这也不难理解，上课时也常会出现本课内容还没全部结束，就被要求去自学剩下的内容的情况呢。

C 没错，这里也是一样。

互动出版网 China-Pub.com 专业图书网上第一专营店 www.china-pub.com 独家提供样章

- **第 15 章（第 15 天）**

Stone 语言的词法分析器由 Java 的正则表达式库实现。第 15 章将不使用这种方式，手工设计词法分析器。具体来讲，这一章将介绍基于正则表达式的字符串匹配程序设计。

- **第 16 章（第 16 天）**

本书采用了解析器组合子库这一简单的库来实现语法分析器。第 16 章将介绍一些语法分析的基本算法，并以 LL 语法分析为基础，手工设计一个简单的语法分析器。

- **第 17 章（第 17 天）**

第 17 章将简单介绍本书使用的解析器组合子库的内部结构，并分析该库的源代码。

- **第 18 章（第 18 天）**

Stone 语言的解释器采用了 GluonJ 系统来实现，该系统允许 Java 语言执行类似于 Ruby 语言中 open class 的功能。第 18 章将总结使用 GluonJ 时的一些琐碎的注意事项。

- **第 19 章（第 19 天）**

抽象语法树是语言处理器的核心。在实现面向对象语言时，抽象语法树的节点对象的类会包含各种类型的方法。本书借助了 GluonJ 来增加这些方法，读者还可以通过其他设计模式来实现相同的效果。第 19 章将介绍使用设计模式实现抽象语法树的优缺点，并与使用 GluonJ 的方式作比较。

A 也就是说全书共有 19 章对吧？老师，那平时时间不多的读者应该优先阅读那些章节比较好呢？

F 你是想问有哪些章节跳过不读也可以对吧？

C 嗯，我建议先读完第 2~8 章，之后是第 15、16 章，如果还有时间，再读一下第 11 章和第 13 章。

F 第 9 章关于面向对象的内容不重要吗？

S 要说最近比较流行的话题，第 14 章的内容才更重要吧。

C 其实如果时间足够，我希望读者能够读完全书。真要选取部分来读的话，我建议按前面讲的顺序阅读。