TURING 图灵程序设计丛书

松本行弘

编程语言的设计与实现

[日] 松本行弘 / 著
[日] 日经Linux / 编
郑明智 / 译

听Ruby之父畅谈

- 设计新语言的动机、过程中的纠结与取舍
- 隐藏在各编程语言背后的设计缘由
- Ruby开发中不为人知的故事

▶ 学习大师级程序员的思维方式

中国工信出版集团
人民邮电出版社
POSTS & TELECOM PRESS

TURING 图灵程序设计丛书

松本行弘编程语言的设计与实现

[日] 松本行弘 / 著
[日] 日经Linux / 编
郑明智 / 译

人民邮电出版社
北 京

图书在版编目(CIP)数据

松本行弘 : 编程语言的设计与实现 / (日) 松本行弘著 ; (日) 日本日经Linux编 ; 郑明智译. -- 北京 : 人民邮电出版社, 2019.8
(图灵程序设计丛书)
ISBN 978-7-115-51616-9

Ⅰ. ①松… Ⅱ. ①松… ②日… ③郑… Ⅲ. ①程序语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2019)第137815号

内 容 提 要

本书由Ruby之父松本行弘在《日经Linux》杂志上的连载整合而成，主要介绍了新语言Streem的设计与实现过程。作者从设计Streem这门新语言的动机开始讲起，由浅入深，详细介绍了新语言开发中的各个环节，以及语言设计上的纠结与取舍，其中也不乏对其他编程语言的调查与思考，向读者展示了创建编程语言的乐趣。

本书适合各层次程序设计人员和编程爱好者阅读。

◆ 著　[日]松本行弘
编　[日]日经Linux
译　郑明智
责任编辑　杜晓静
责任印制　周昇亮
◆ 人民邮电出版社出版发行　北京市丰台区成寿寺路11号
邮编　100164　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
大厂聚鑫印刷有限责任公司印刷
◆ 开本：800×1000　1/16
印张：20.25
字数：474千字　2019年8月第1版
印数：1-3 500册　2019年8月河北第1次印刷
著作权合同登记号　图字：01-2017-3629号

定价：89.00元

读者服务热线：(010)51095183转600　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字20170147号

译者序

本书是 Ruby 之父松本行弘最新的作品。在书中作者讲述了从头开始设计一门新的流处理语言 Streem 的过程。凭借作者在业界的影响力，Streem 语言在开发阶段就已经在 GitHub 上收获了四千多 star。

虽然 Streem 语言依然在开发阶段，不适合在正式项目中使用，但是本书最大的价值并不在于 Streem 语言本身。正如作者在文中所说，很多编程语言相关的书都是在介绍一门语言该如何使用，自制编程语言的书也仅仅涉及语言的实现，讲解语言为什么这样设计的书少之又少，而这本书就属于后者。从简单的如何为语言取名，到复杂的如何设计多线程、垃圾回收等，作者在书中事无巨细地介绍了语言设计的方方面面。对于想了解语言为什么会这样设计、语法为什么是这样的读者来说，本书是一本不可多得的宝贵资料。

练武的有武痴，而作者简直就是“语言痴”（或者“语言控”，褒义），几十年来一直孜孜不倦地研究各种语言，并且与许多语言设计者有过直接的交流，所以对很多语言都很熟悉，在书中他也旁征博引地介绍了这些语言是如何设计的。因为很多语言的语法都是相通的，所以即使不打算自己去设计新的语言，了解语言设计的知识，对语言的学习也是大有裨益的。比如 3.4 节讲解了模式匹配，对于 Java 程序员理解 Scala 语言的模式匹配功能应该很有帮助。4.4 节综合介绍了垃圾回收的几种算法，写得通俗易懂，与 Java 或者 Objective-C 等语法书一般只介绍语言中用到的垃圾回收算法相比，作者的介绍更有助于读者深入理解垃圾回收，了解各种算法的优缺点，这也许在面试时就能用得上。

作者在业界活跃几十年，学识非常渊博。第 5 章就是一个很好的体现。作者在第 5 章讲解了管道编程、CSV 文件处理、时间表示、统计以及随机数等多个主题，并围绕这些主题介绍了大量相关的知识和算法。比如 5.2 节介绍了流处理相关的 `map`、`reduce` 和 `flatmap` 等函数，这些知识对学习大数据的 mapreduce 和 spark 流处理会很有帮助；5.5 节介绍的统计算法和 5.6 节介绍的随机数生成算法让译者在翻译时也受益匪浅。这些知识说不定什么时候就能派上用场，如果你记住了 5.5 节介绍的蓄水池抽样算法，那么海量数据随机抽样之类的问题就难不倒你了。

虽然作者是业界的风云人物，但在书中毫不讳言自己不擅长哪些领域、哪些地方借鉴了他人的实现，非常平易近人，不会让人感觉高不可攀。阅读本书就像是在跟一位谆谆善诱的前辈聊天，他会告诉你自己是怎么设计语言的、为什么要这样设计、别人是怎么做的、具体要怎么实现、在自己实现不了的情况下如何借鉴别人开源的东西、如何取得许可，等等，非常有趣。作者还在书中设置了“时光机专栏”，其中添加了对正文内容经过一段时间沉淀之后的思考，这些内容也很有趣，请读者不要错过。

另外，作者在书中多次流露了“最近太忙了，这个功能有机会再实现吧”的想法，让译者也心有戚戚焉。

希望读者能在阅读本书的过程中体验到语言设计的乐趣，有所收获。

由于译者水平有限，书中恐有疏漏和错误之处，还请读者随时指正。

在此衷心感谢图灵公司的各位编辑在翻译过程中给予的帮助，感谢博学儒雅的上司王蕴韬在工作上的指导，感谢亲友杨玉生、黄旭、戎政给予的支持和鼓励，最后也要感谢一直给我家庭温暖的父母、岳父母、妹妹一家、妻子和儿子。

郑明智

2019 年 5 月于余杭南湖

前言

我曾经在 Linux 专业期刊《日经 Linux》上开设了名为“边做边学编程语言”的专栏（2014 年 4 月 ~ 2016 年 12 月），本书就是对这个专栏的系列文章加以汇总，并添加了一些新的内容编辑整理而成的。

以自制编程语言为主题的书非常多，我家的书架上也放了好几本。这些自制编程语言的书绝大部分是介绍编程语言的实现的。比如以比较简单的语言的实现作为示例，介绍如何使用 yacc 和 lex 工具制作语法分析器和词法分析器，如何实现解释器等。

不了解真实情况的人总以为编程语言的实现需要高超的技术，非常困难，但如果循序渐进地进行实际操作，就会发现其实并没有那么难。实际上，在大学的计算机科学课程中，也有不少实现语言处理器之类的作业。在以年轻人为对象的编程竞赛中，比如我长期担任评委的 U22 编程竞赛，甚至有高中生挑战自制编程语言。

抛开成见认真去做的话，你会发现编程语言的设计与实现对程序员来说是一项非常有趣的智力挑战。在这个意义上，这些关于自制编程语言的书是有其存在价值的。

但这并不是说我对这些书没有不满。这些书顶多是对编程语言的实现的解说，作为示例使用的语言也几乎都是现有语言的（过于简化的）子集。而对于创建编程语言的过程中最挑战智力也最有趣的语言设计部分，可以说完全没有涉及。

不过这也是没办法的事，可以理解。为了有效利用有限的页数，缩小主题是必然的。如果不限定为简单的语法，那么无论如何也很难讲完。再往深里说，也没有多少人有设计全新的编程语言的经验，更不用说自己设计的语言在全世界被广泛使用了，可以毫不夸张地说，根本没有这样的人。

也就是说，几乎没有人能根据自己的经验来讲述如何设计编程语言。因此，将语言的设计作为后话，优先讲解语言处理器的实现技术，可以说是必然的。

其中也有少数例外的情况，比如 C++ 设计者本贾尼·斯特劳斯特卢普（Bjarne Stroustrup）所著的 *The Design and Evolution of C++*[①] 一书。这是一本非常宝贵的书。读了这本书，你就可以了解 C++ 为何是现在的样子，它的目标是什么。不过是否能靠这本书学会语言设计的方法，那就另当别论了。

既然世界上不存在这样的书，那就只能自己去写了。幸好我拥有在全世界被广泛使用的编程语言的设计经验，有这种经验的人屈指可数。另外，我兴趣广泛，对世界上各种编程语言的设计都有所了解。再者，我在《日经 Linux》上连载过文章，还有多本书的写作经验。因此，要写关于设计编程语言的书，恐怕整个日本没有人比我更合适了吧（自吹自擂）！

有了这个想法之后，我便从《日经 Linux》2014 年 4 月刊起开设了专栏，开始发表该主题的文

① 中文版名为《C++ 语言的设计与演化》，裘宗燕译，科学出版社 2012 年出版。——译者注

章（表 1）。

从创造编程语言的动机，到思考语言设计时的内心纠葛……本书提到了很多其他同类书中不曾涉及的内容，对此我也颇为骄傲。

但是在把每个月连载在杂志上的内容汇总成书时，一个难逃的宿命就是，随着时间的推移，会出现内容前后矛盾等问题。本书也不例外。

连载时的主题如表 1 所示。其中，嵌入式 Ruby“mruby”相关的介绍（2014 年 4 月刊和 2014 年 6 月刊的一部分）以及超出本书范围的相关说明（2014 年 9 月刊～2014 年 11 月刊）均未被收录在本书之中。另外，我还调整了连载的前后顺序，并结合 Streem 的现状修改了部分内容。

但是，整体的文章结构与连载时相比并没有大的变动。因此，为了保持内容的连贯性以及帮助各位读者理解，我在每节的末尾添加了“时光机专栏”，这是对正文的补充。在这部分内容中，我会站在现在的角度审视过去的连载主题有哪些不足之处。

“时光机专栏”的存在，好像是向世人承认自己的能力有限一样，这令我心情复杂，但想到谁也无法预知未来，我也就释然了。

接下来就由我带领大家进入编程语言设计的世界吧！

松本行弘

2016 年 12 月

表 1　连载主题一览表

《日经 Linux》刊登期号	标题	本书
2014 年 4 月刊	第 1 期　编程语言设计入门	1-1
2014 年 5 月刊	第 2 期　语言处理器的结构	1-2
2014 年 6 月刊	第 3 期　虚拟机	1-3
2014 年 7 月刊	第 4 期　编程语言设计入门（前篇）	1-4
2014 年 8 月刊	第 5 期　编程语言设计入门（后篇）	1-5
2014 年 9 月刊	第 6 期　编程语言的类型设计 (1)	-
2014 年 10 月刊	第 7 期　编程语言的类型设计 (2)	-
2014 年 11 月刊	第 8 期　编程语言的类型设计 (3)	-
2014 年 12 月刊	第 9 期　抽象的并发编程	2-1
2015 年 1 月刊	第 10 期　21 世纪的并发语言	2-2
2015 年 2 月刊	第 11 期　设计 Streem 语言	2-3
2015 年 3 月刊	第 12 期　Streem 语言的核心	2-4
2015 年 4 月刊	第 13 期　多线程与对象	2-5
2015 年 5 月刊	第 14 期　缓存与符号	2-6
2015 年 6 月刊	第 15 期　AST	2-7
2015 年 7 月刊	第 16 期　局部变量与异常处理	2-8
2015 年 8 月刊	第 17 期　套接字编程	4-1
2015 年 9 月刊	第 18 期　CSV	5-3
2015 年 10 月刊	第 19 期　基本数据结构	4-2
2015 年 11 月刊	第 20 期　各种各样的面向对象	3-1
2015 年 12 月刊	第 21 期　Streem 的面向对象	3-2
2016 年 1 月刊	第 22 期　对象表示与 NaN Boxing	4-3
2016 年 2 月刊	第 23 期　垃圾回收	4-4
2016 年 3 月刊	第 24 期　管道编程	5-1
2016 年 4 月刊	第 25 期　管道的组成要素	5-2
2016 年 5 月刊	第 26 期　无锁算法	4-5
2016 年 6 月刊	第 27 期　时间表示	5-4
2016 年 7 月刊	第 28 期　统计基础	5-5
2016 年 8 月刊	第 29 期　再看 Streem 的语法	3-3
2016 年 9 月刊	第 30 期　模式匹配	3-4
2016 年 10 月刊	第 31 期　随机数	5-6
2016 年 11 月刊	第 32 期　数据流图	5-7
2016 年 12 月刊	最后一期　后记与拾遗	-

目录

第 1 章

创造一门什么样的语言

1-1 自己创造编程语言的意义

通过实际创造一门新的编程语言，可以学到编程语言的设计思路和实现方法。随着开源的普及，创造新编程语言的门槛一下子降低了许多。创造编程语言不仅可以提升你作为技术者的价值，而且还可以使你从中获得很大的乐趣。

大家都知道我是编程语言 Ruby 的作者，我其实还是一个编程语言迷，对编程语言的痴迷程度无人能及。Ruby 是我出于兴趣钻研编程语言的最大成果，把它称为我兴趣的副产品可能更为贴切。副产品就能如此普及看起来很了不起，但与其把它全部归功于我的实力，倒不如说运气的成分更大。Ruby 已经诞生 20 多年了，如果没有这么多年来发生的各种事情与邂逅，根本不可能有今天这样的成绩。

进入创造编程语言的世界

大家有创造编程语言的经历吗？对于有过编程经历的人来说，编程语言是非常亲切的存在，但是他们往往会认为编程语言是现成的东西，也许谁都没有想过自己去创造一门新的编程语言。这也是情理之中的事情。

与人们说话用的语言（自然语言）不同，世界上所有的编程语言都是由某个地方的某个人创造的。它们不是自然产生的，而是根据明确的意图和目的被设计并实现的。所以，如果过去没有这些创造编程语言的人（编程语言的作者），那么我们今天可能还在用汇编语言编程呢。

在人们刚开始编程时，编程语言就随之出现了，可以说编程的历史就是编程语言的历史。

本书的目标是要你自己创造一门编程语言。可能有的读者会想："现在再创造编程语言还有什么意义呢？"我稍后回答这个问题，现在我们先来看一下编程语言的历史。

■ 个人创造编程语言的历史

早期的编程语言是由在工作中切切实实与编程语言打交道的人创造的，这些人大多就职于企业的研究所（比如 FORTRAN、PL/1 的发明）、大学（比如 LISP）以及标准委员会（比如 ALGOL、COBOL）等。也就是说，设计开发编程语言是专业人士的工作，但是这个传统随着 20 世纪 70 年代计算机的普及开始发生了变化。一些计算机爱好者在拥有了自己的计算机后，出于兴趣开始编程，甚至开始开发新的编程语言。

其中最具有代表性的就是 BASIC 语言。BASIC 语言原本是美国达特茅斯学院用于教学的编程

语言，它的语法非常简单，用极少的代码实现了最基本的功能，所以深受 20 世纪 70 年代编程爱好者的喜爱，并被他们广泛使用。

这些编程爱好者也开始开发自己版本的 BASIC 语言。当时，个人计算机[①]的内存顶多几千兆，他们开发的 BASIC 语言就是可以在内存如此之小的机器上工作的小规模版本。这些小规模的 BASIC 程序大小不到 1 KB，它们在 4 KB 左右的内存上也能工作，跟现在需要大内存的语言处理器比起来真是令人惊讶。

微机杂志的时代

以个人开发的 BASIC 为代表的小规模语言（Tiny 语言）处理器不久便以各种各样的形式进行了发布。当时的软件有的以 Dump list 的形式刊登在计算机杂志上，有的将程序数据进行音频转换后收录在杂志附带的薄膜唱片（sonosheet）中发布。现在的人恐怕已经不知道薄膜唱片了吧。薄膜唱片是指塑料做的薄薄的唱片，不过唱片这个词几乎没有人用了。据说当时的计算机爱好者都用唱片播放器连接计算机来读取数据，而不使用磁带录音机这个最普遍的外部存储设备。

20 世纪七八十年代是计算机杂志（当时称为微机杂志）的全盛时期，在日本以下 4 种杂志竞争激烈。

- *RAM*（广济堂出版）
- *My Computer*（电波新闻社）
- *I/O*（工学社）
- *ASCII*（ASCII 公司）

这 4 种杂志中现在只有 *I/O* 仍在发行，不过也大不如前了。作为一个了解当时情况的人，我的内心充满了无限感慨。

这之后，*My Computer* 杂志派生出了 *My Computer BASIC Magazine*，又发生了很多事情，继续讲下去恐怕就会变成上岁数人的叙旧了，所以点到为止吧。如果去问问现在三四十岁的程序员，相信他们中间很多人都会眉飞色舞地讲起那个年代的事情。

当时的微机杂志附带了收录 BASIC 的薄膜唱片，除此之外还介绍了其他几个小规模语言，如 GAME、TL/1 等。这些语言都反映了当时那个时代的特色，非常有趣，我会在本节的最后对其进行介绍，请大家务必读一读。

个人创造编程语言的现状

为什么从 20 世纪 70 年代后期到 80 年代前期开始兴起个人创造编程语言了呢？我认为最大的

① 通常称为微机。微机是微型计算机、微型机的简称。

原因是当时难以获取开发环境。

20 世纪 70 年代后期广泛使用的微机是 TK-80（图 1-1）那样的主板裸露在外的单板机，很多都是半成品，需要自己去钎焊。这样的机器不可能自带开发环境之类的东西，软件都要自己输入机器语言之后才会工作。

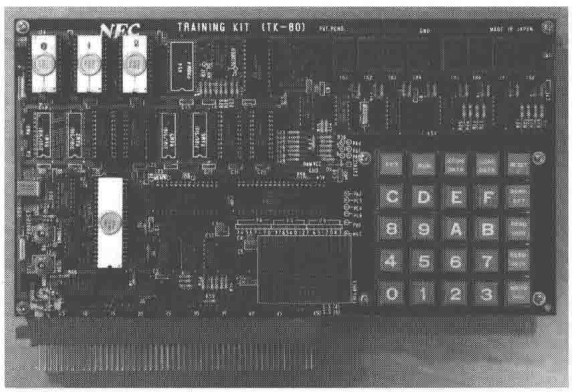

图 1-1 TK-80

20 世纪 70 年代末期才出现 PC-8001 和 MZ-80 那样的“成品计算机”。然而，这种计算机顶多带一个 BASIC 开发环境，因此人们很难自由地选择开发语言。虽说市面上也有商用的语言处理器，但 C 编译器的定价就要 19.8 万日元，这不是普通人可以轻易买得起的。于是，人们便有了热情去创造一门自己的编程语言。

可现在获取语言的开发环境已经不再是麻烦事了。各种编程语言和开发环境作为开源软件被公开，即使是非开源的，也可以轻松地通过网络得到免费版本。

这样一来，现在自己创造编程语言岂不是没有任何意义吗？如果这个问题的答案为“是”，那么本书在第 1 章开头就结束了。

我认为（而且为了这本书也应当这么回答），这个问题的答案为“否”。即使是现在，自己创造一门新的编程语言也是有意义的，而且有很重要的意义。

而且现在很多广泛使用的编程语言也都是在开发环境容易获取的情况下，由个人设计和开发出来的。如果个人开发编程语言真的没有意义，那么 Ruby、Perl、Python 和 Clojure 这些语言也就不会诞生了。

不过即便如此，我认为 Java、JavaScript、Erlang 和 Haskell 这些语言也可能会以其他形式出现，因为它们会作为业务和研究的一环被开发出来。

■ 为什么要创造新的编程语言

那么如今个人设计开发编程语言的动力究竟是什么呢？

回顾我自身的经历以及参考其他语言作者的意见，我认为有以下几点理由。

- 提高编程能力
- 提高设计能力
- 打造个人品牌
- 获得自由

首先，编程语言的实现可以说是计算机科学的综合艺术。作为语言处理器的基础，词法分析和语法分析也可以应用在网络通信的数据协议的实现等方面。

实现语言功能的库和实现其中的数据结构，这正是计算机科学要做的事情。尤其是编程语言的应用范围广泛，很难事先预测会被用于什么方面，因此库和数据结构的实现难度也就更大，但也变得更加有意思了。

另外，编程语言还是人与计算机间的接口。设计这样的接口，就需要深入考察人是如何思考问题的、下意识中有什么样的期待。反复进行这样的考察，对编程语言之外的应用程序接口（API）设计、用户界面（UI）设计，甚至用户体验（UX）设计都是有益的。

提升个人品牌

也许有人会感到意外，实际上在 IT 行业，对编程语言感兴趣的人不在少数。这是毋庸置疑的，因为编程与编程语言有着切不断的关系。以编程语言为主题的活动和会议等往往都会吸引很多人参加，由此我们也能感受到编程语言的魅力。正因如此，很多人在网上发现新的语言后就会开始尝试。就拿 Ruby 来说，它在 1995 年被发布到网上之后，仅仅 2 周左右就吸引了 200 多人加入邮件列表，着实令人惊讶。

可是，虽然有很多人愿意尝试使用新的编程语言，却几乎没有人会去设计并实现一门编程语言，而且是超越杂志提及的“小儿科语言”那种程度的能够实用化的编程语言。但我保证，仅凭设计出一个实用的编程语言这一点，你就会得到人们的尊敬。

在这个开源的时代，技术人要想生存下去，在技术社区的存在感是非常重要的。虽然技术人只要开源其软件就能达到站稳脚跟的效果，但编程语言的“特殊感”会进一步提升其品牌效应。

乐趣第一

另外，编程语言的设计与实现比任何事情都更有趣。的确如此。与计算机科学相关的具有挑战性的工程也是这样。设计编程语言还可以帮助使用这门语言的程序员思考，甚至左右他们的想法，这一点也非常有意思。

通常来说，编程语言有一种从别处获取的、不容侵犯的感觉。如果是自己创造编程语言，就完全没有这个问题。你可以按照自己的喜好进行设计，如果不满意或者有更好的想法，也可以自由地修改。从某种意义上来说，这是终极的自由。

编程在某种意义上是对自由的追求。通过亲自编程，我们可以获得单纯使用他人的软件时享受不到的自由。至少对我来说，这是编程的一个重要动机。于我而言，创造编程语言是获取更高程度自由的手段，也是我的乐趣与快乐的源泉。

为什么创造新编程语言的人不多

虽说自己创造一门编程语言有这么多好处，但并不是每个人都会去做。正如上文所说的那样，对编程语言感兴趣的人虽然有一些，但着手去创造编程语言的人几乎没有。说是“感兴趣的人有一些”，但从占总人口的比例来看，其实少到可以算作误差范围的程度，更不用说有动力去创造新编程语言的人了，就算没有也不足为奇。

我自己在关注编程语言几年后就着了迷，但是在进入大学主修计算机科学之后，才注意到并不是所有人都对编程语言感兴趣。这是因为我在偏僻的乡下长大，周围没有喜欢编程的人可供比较。这一点对我来说也不知道是幸还是不幸。

“难道我跟别人不一样?”意识到这一点的时候，我很震惊。因为当时的微机杂志上刊登了很多关于 TL/1 等编程语言的文章。我本以为对编程感兴趣的人（和我一样）很可能也会对编程语言着迷，但实际上并非如此。

本来就对编程语言不感兴趣的人自不用说，即使是感兴趣的人，也很难走到自己设计并实现编程语言这一步。

关于这个问题的原因，我思考过很长时间。作为编程语言设计者，在参加编程语言相关的活动时，我也曾以过来人的身份鼓励别人尝试一下，但结果总是不尽如人意。当然，万事开头难，开始一件新的事情是需要很大勇气的。但即使是这样，反响也太差了。

没必要想得很难

问了很多人之后，我才知道大家为什么不去着手尝试了。那是因为就算有兴趣创造一门新的编程语言，在开始之前多半也会有某种心理障碍，也就是觉得“编程语言有现成的，本来就不需要自己去设计和开发”。难得有那么几个人不会产生这种心理障碍，却又觉得语言的实现似乎很难。也就是说，他们觉得编程语言很有趣，自己也想做做看，却不知道如何去实现。

仔细想来，关于编程语言的实现的书虽然出乎意料地出版了很多，但大部分都是大学教材的难度，非常不容易理解。另外，与编译原理有关的“文法类型”和“Follow 集合”等晦涩的术语也频繁出现。

但是认真想一想，我们的目的是出于兴趣创造自己的编程语言，而不是去掌握编程语言的实现所需的所有知识。如果你认为在没有完全掌握正确的知识之前就无法着手创造编程语言，那就大错特错了，你的热情会被逐渐消磨殆尽。

成就一番伟大的事业首先需要的就是热情，不能保持热情是不行的。一旦有了创造编程语言的热情，就应尽快开始，以后再根据需要慢慢地掌握所需的知识即可。

本书主要介绍创造简单的语言处理器所需要的基本知识以及工具的使用方法，并不涉及编程语言实现的较难部分。相较于理论背景，我更想把重点放在如何设计编程语言上。

微机杂志中介绍的Tiny语言

GAME

GAME（General Algorithmic Micro Expressions）是由BASIC派生的Tiny语言。它最大的特征是关键字全部是符号，以及所有的语句都是赋值语句。

例如赋值给“?”时会输出数值，反过来将“?”赋值给变量时会要求输入数值。字符串的输入输出使用“$”。另外，将行号赋给“#”时为goto语句，将行号赋给“!”时为gosub语句（调用子程序）。

另外，像"ABC"这样的一个字符串语句会打印出字符串，后面有“/”的话会换行。

这是一门非常有意思的编程语言，示例代码如图1-A所示。它既像BASIC，又不像BASIC，请大家好好感受一下。

GAME是一门非常简洁的语言，用8080汇编语言编写的解释器的大小还不到1 KB。另外，由中岛聪（当时居然还是高中生）开发的使用GAME编写的GAME编译器，代码仅有200行左右。真不知道我们应该惊叹GAME的语言表现能力，还是中岛聪的技术能力。

TL/1

同一时期在*ASCII*杂志上发表的Tiny语言中还有TL/1（Tiny Language/1），它的名字应该是模仿了美国IBM公司开发的编程语言PL/1。与受BASIC影响使用符号的GAME语言不同，TL/1拥有类似于Pascal的语法，让人觉得更加“正常”。另外，TL/1的语言处理器是编译型的，与主体为解释型的GAME比起来速度更快。但实际上GAME也有编译器，这一点我们在前文中介绍过。

TL/1的特征是语法类似于Pascal，以及变量类型只有1字节的整数。各位读者也许会想这样怎么编写代码，不过当时的主流CPU是8位的，所以TL/1设计成这样也不是很怪异。虽说是运行在8位CPU上，但包括GAME在内的其他语言都提供了16位的整数类型。

那么1字节无法表示的超过255的数值该如何编写呢？答案是按字节进行分割，用多个变量组合表示。比如用2个变量保存16位整数，边看计算溢出时的进位标志边计算（图1-Ba）。在当时的8位CPU上，大部分处理用16位整数就已经足够了（地址用16位的话就可以访问所有地址空间）。作为Tiny语言，这样的功能已经足够了。各位读者如果有兴趣，可以显式地查看进位标志，使用多个变量进行24位计算或32位计算。

可以处理指针和字符串

另外，指针也无法仅用1字节来表示。这里用mem数组进行访问，也就是说，用下面表

达式中hi,lo表示的16位地址来访问指定地址的内容。

```
mem(hi, lo)
```

下面是将地址的值替换为v。

```
mem(hi, lo) = v
```

当时的个人计算机（微机）最多只有32 KB的内存，所以能用16位地址访问就已经足够了。

还有就是字符串。当然，我们也可以将字符串当作字节数组，对每个字节依次进行操作，但是这样处理太麻烦，因此TL/1设计了用于数据输出的WRITE语句。

例如，用TL/1开发的Hello World程序如图1-Bb所示。TL/1中变量本应只有1字节的整数，却出现了字符串。实际上，WRITE是为了能够处理字符串而单独增加的语法。

WRITE之外的语句是无法处理字符串的，所以不能进行普通的字符串处理，只能够操作1字节的整数。现在看来可能会觉得很不可思议，但是在不属于Tiny语言的Pascal和FORTRAN中，输入输出也是被特殊处理的，这在当时也许是一种比较普遍的做法。

```
100---------------- Comment -------------------
110|    如果紧接在行号之后的不是空白，则将该行作为注释
120-------------------------------------------
130
200 / " FOR循环语句 是 变量名=初始值，最终值  ... @=(变量名 + 步长) " /
210  A=1,10
220    ?(6)=A
230  @=(A+1)
240
300 / " IF语句的例子 " /
310  B=1,2
320    ;=B=1 " B=1 " /
330    ;=B=2 " B=2 " /
340  @=(B+1)
350
400 / " 数值输入与计算 " /
410  "A = ?" A=?
410  "B = ?" B=?
420  "A + B = " ?=A " + " ?=B " = " ?=A+B /
```