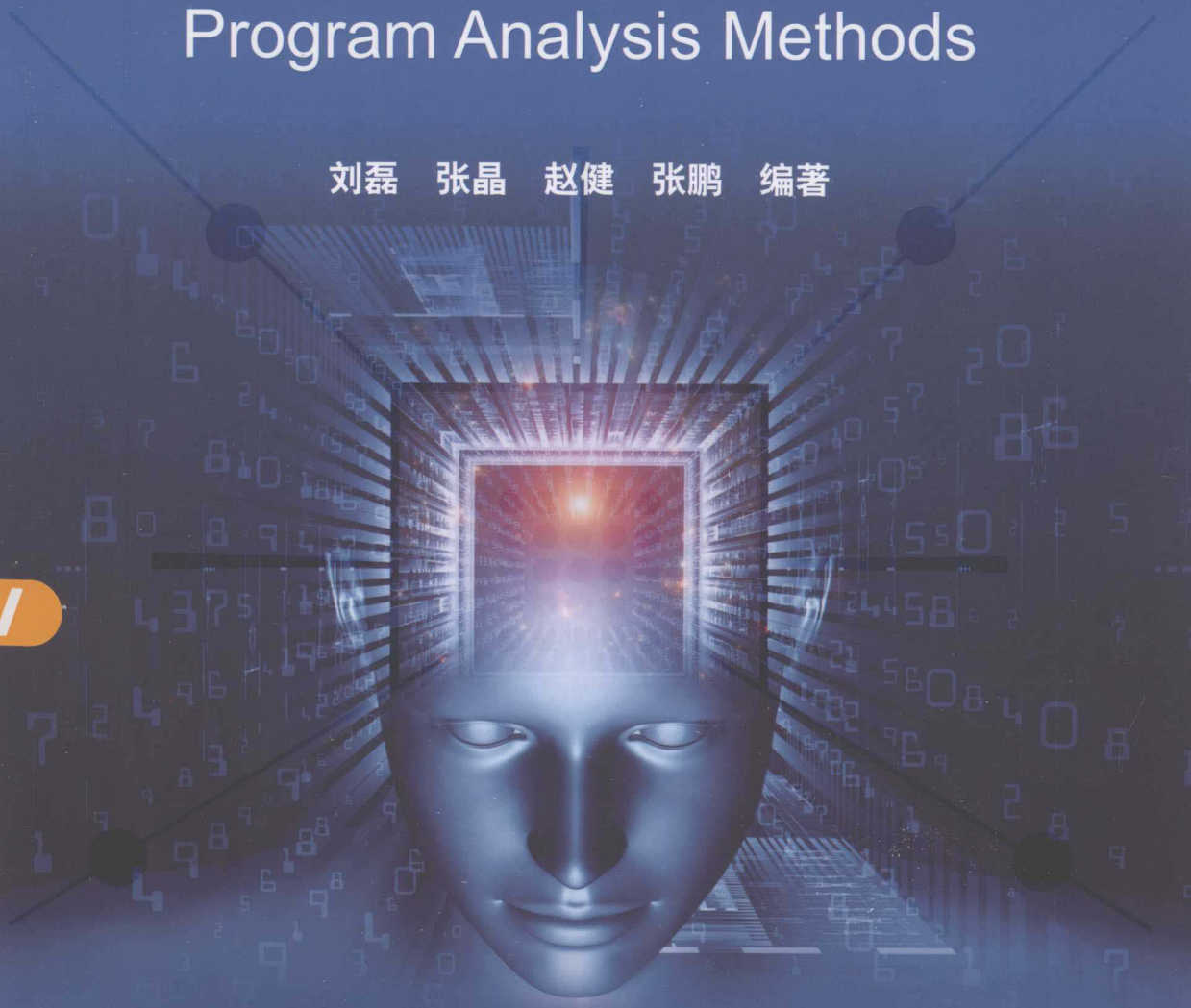
HZ BOOKS
华章科技

国内首本程序分析方法著作，资深专家撰写，权威性毋庸置疑。
本书全面系统地讲解了诸多实用的程序分析方法。

程序分析方法

Program Analysis Methods

刘磊　张晶　赵健　张鹏　编著



机械工业出版社
China Machine Press

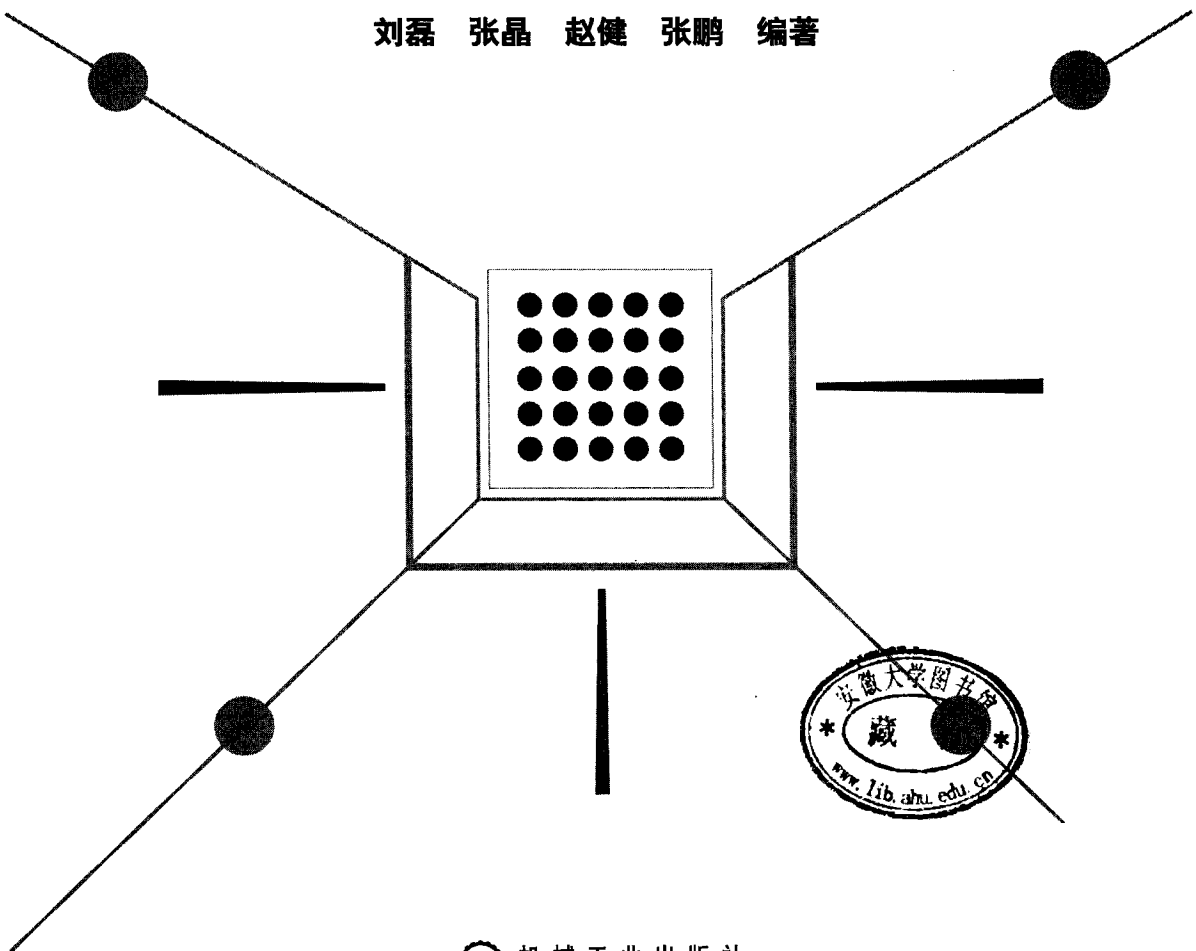
国内首本程序分析方法著作，资深专家撰写，权威性毋庸置疑

本书全面系统地讲解了诸多实用的程序分析方法

程序分析方法

Program Analysis Methods

刘磊　张晶　赵健　张鹏　编著



机械工业出版社
China Machine Press

图书在版编目（CIP）数据

程序分析方法/刘磊等编著. —北京：机械工业出版社，2013.4

ISBN 978-7-111-42252-5

Ⅰ. 程… Ⅱ. 刘… Ⅲ. 程序分析 Ⅳ. TP311.11

中国版本图书馆 CIP 数据核字（2013）第 083152 号

本书共 7 章，第 1 章介绍程序设计语言的历史以及未来的发展趋势；第 2 章介绍元程序设计，包括元程序的概述、元程序系统及其应用；第 3 章介绍信息流分析技术；第 4 章介绍别名分析的相关知识；第 5 章介绍程序分片的基础知识和方法；第 6 章介绍形式概念分析的相关知识和应用；第 7 章介绍部分求值技术的相关内容。

本书可作为计算机相关专业本科高年级学生及研究生教材，也可作为程序开发人员的参考用书。

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）

责任编辑：王 彬

北京京师印务有限公司印刷

2013 年 6 月第 1 版第 1 次印刷

186mm×240mm • 12.75 印张

标准书号：ISBN 978-7-111-42252-5

定 价：45.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

客服热线：（010）88378991 88361066 投稿热线：（010）88379604

购书热线：（010）68326294 88379649 68995259 读者信箱：hzjsj@hzbook.com

PREFACE 前言

程序分析是以某种语言编写的程序为对象，对其内部的运作流程进行分析的过程。通过程序分析，可以帮助人们更好地理解程序内部各模块之间的调用关系，把握程序的运行流程，也可以帮助人们找出系统运行的瓶颈，指导系统优化，还可以用于系统测试和程序调试，找出系统 Bug，以及进行错误定位。因此，程序分析是一种非常实用的技术，深入理解并熟练掌握程序分析相关的方法和技术，对于增强对程序设计语言的理解，掌握先进的程序设计方法，更好地分析和理解程序都有很大帮助。程序分析是计算机专业人员必备的一种技术，同时也应该成为计算机专业、软件工程专业学生（本科生、研究生）今后进行科学研究的专业必修课。

本书作者多年来一直为吉林大学计算机专业、软件工程专业的研究生讲授程序分析的课程，因内容广泛、方法实用，该课程深受学生欢迎。遗憾的是，这门课程一直缺少相应的教材，国内这方面的参考书也非常少。鉴于此，本书作者根据自己多年的教学和科研经验，在以往研究生授课讲义的基础上编写了此书。本书旨在向读者介绍程序分析的各种方法、技巧。

程序分析包含的范围非常广泛，考虑到教学方便和学生需求，作者尽可能地选取那些比较实用、应用范围较广、内容跟得上时代发展的程序分析方法。本书选取的程序分析方法有：元程序设计、数据流分析、控制流分析、部分求值、形式概念分析、程序分片、程序转换等。程序是与程序设计语言紧密相关的，为此，作者又特别增加了一章，即程序设计语言部分。

本书具有如下特点：

1）在内容安排上，每一章介绍一个专题，每个专题自成体系，读者可以根据需要选取感兴趣的专题进行阅读。根据专题内容的不同，介绍的侧重点也各有不同：有的侧重原理的讲解、有的侧重应用实例的分析、有的侧重算法的设计，尽量做到重点突出、讲解透彻，非常适合本科生高年级和研究生阶段的学习特点。

2）对于书中的许多专题，在介绍基本内容的同时，又融入了我们自己的许多科研成果，如元程序设计专题中的面向对象元程序设计方法；部分求值专题中的过程式语言

的动态部分求值和静态部分求值技术；程序分片专题中的过程间的程序分片技术等。

3）我们力争把最新的技术和方法介绍给广大的读者，如近年来比较热门的形式概念分析技术等。

4）将程序分析技术的几方面的知识合为一体，读者可以通过本书学习到多方面的内容，免去了查找相关资料的烦恼。

金成植对于本书的编写给予了大力支持和帮助，刘华虓、张鑫、刘冬清、郭骐凯等对全书进行了校对，在此向他们表示衷心的感谢。

由于作者水平和时间限制，书中难免存在疏漏和不足之处，恳请广大读者批评指正。

作 者

2013 年 2 月

CONTENTS 目 录

CHAPTER 1

第 1 章　程序设计语言

自然语言是人与人之间沟通和交流的工具，而程序设计语言（Programming Language）是程序员与计算机或程序员与程序员之间沟通和交流的工具。语言的基本组成是一组符号和一组规则，依据规则由符号构成的符号串的总体就是语言。在程序设计语言中，这些记号串就是计算机程序（Program）。一般来说，计算机程序就是用某种程序设计语言编写的一个指令序列，用于说明使用计算机将完成的任务的工作流程或工作步骤。计算机的控制器从程序的第一条指令开始，顺序地逐条取出指令，然后按指令的规定和要求“指挥”整个计算机系统的工作，从而完成人们设想的要计算机完成的工作。计算机的使用者以计算机程序的形式向计算机提出服务请求，计算机按程序自动进行工作，是计算机系统最基本的原理。为计算机编写程序的过程称为程序设计（Programming）。

程序设计语言包含三方面的要素，即语法、语义和语用。语法表示程序的结构或形式，亦即表示构成语言的各个记号之间的组合规律，但不涉及这些记号的特定含义，也不涉及使用者。语义表示程序的含义，亦即表示按照各种方法所表示的各个记号的特定含义，但不涉及使用者。语用表示程序与使用者的关系。

对于从事与计算机科学相关的人来说，理解程序设计语言是十分必要的，因为当今所有的计算任务都需要利用程序设计语言编写的计算机程序才能完成。从第一台电子计算机诞生到现在，大量的程序设计语言被发明、被取代、被修改或被组合在一起。尽管人们多次试图创造一种通用的程序设计语言，却没有一次尝试是成功的。

1.1　程序设计语言发展的四个阶段

到目前为止，程序设计语言的发展经过了机器语言、汇编语言、高级语言、第四代语言四个阶段，每一个阶段都使程序设计的效率大大提高。我们常常把机器语言称为第

一代程序设计语言，把汇编语言称为第二代程序设计语言，把高级语言称为第三代程序设计语言，把最新的程序设计语言称为第四代语言。

1.1.1 机器语言

机器语言是计算机能直接识别和执行的一组机器指令的集合。它是计算机的设计者通过计算机的硬件结构赋予计算机的操作功能。一条机器指令就是机器语言的一个语句，它是一组有意义的二进制代码。每条机器指令一般由操作码和地址码两部分构成，其中操作码说明指令的含义，地址码说明操作数的地址。机器语言程序能够在对应型号的计算机上直接运行。

【例 1.1】 若计算：

$$Y=\begin{cases}X+15 & \text{若}X<Y\\ X-15 & \text{若}X\geqslant Y\end{cases}$$

用 Pentium 机器语言可编出如下程序片段（设程序从 100 号单元开始；X、Y 分别占用 116、118 号单元）。

```
1010 1001 0001 0110 0000 0001
0011 1100 0001 1000 0000 0001
0111 1100 0000 0101
0010 1101 0001 0101 0000 0000
1110 1010 0000 0011
0000 0101 0001 0101 0000 0000
1010 0011 0001 1000 0000 0001
…… …… …… …… …… …… ……
0000 0000 0000 0000
0000 0000 0000 0000
```

从上面的例子可以看出：

- 用机器语言进行程序设计比较烦琐。首先，程序员要熟记所用计算机的全部指令集及每条指令的含义；其次，在程序编写过程中，程序员要自己处理每条指令和每一数据的存储分配和输入/输出，还要记住每步中所使用的工作单元处于何种状态。由于程序员既要驾驭程序全局又要深入每个局部细节，因此程序的开发周期长、可靠性差。
- 机器语言编写出的程序都是由 0 和 1 构成的符号串，可读性差，还容易出错，不易交流和维护。
- 机器语言编程的思维及表达方式与程序员日常的思维和表达方式差距较大，程序员需要经过长期的训练才能胜任。

- 机器语言程序设计严重依赖于具体计算机的指令集，编写出的程序可移植性差、重用性差。

基于上述原因，人们引进了汇编语言。

1.1.2　汇编语言

鉴于机器语言编程的烦琐，为减小程序员在编程中的劳动强度，20 世纪 50 年代中期，人们开始用一些“助记符号”来代替 0、1 码编程，即用助记符代替机器指令中的操作码，用地址符号或标号代替机器指令中的地址码，将机器语言变成了汇编语言。汇编语言也称符号语言，即符号化的机器语言，提高了程序的可读性和程序开发效率。

完成例 1.1 中同样的计算，Pentium 汇编语言程序片段如下：

```
    MOV AX  ,X
    CMP AX  ,Y
    JL      S1
    SUB     AX  ,15
    JMP     S2
S1:ADD AX   ,15
S2:MOV Y    ,AX
    … … … … … …
    X  DW  ?
    Y  DW  ?
```

汇编语言用助记符而不是 0 和 1 序列来表示指令，程序的生产效率和质量都有所提高。但是使用汇编语言编写的程序，计算机不能直接识别，必须有一种程序将汇编语言翻译成机器语言，起这种翻译作用的程序称为汇编程序（Assembler），汇编程序把汇编语言翻译成机器语言的过程称为汇编（Assembling）。

汇编语言程序经汇编得到的目标程序占用内存空间少，运行速度快，有着高级语言不可替代的作用，因此汇编语言常用来编写系统软件和过程控制软件。

汇编语言和机器语言都与具体的机器有关，它们都称为面向机器的语言，也称为低级语言。程序员用它们编程时，不仅要考虑解题思路，还要熟悉机器的内部构造，并且要“手工”地进行存储器分配，编程的劳动强度仍然很大，这些仍然阻碍着计算机的普及和推广。因此，人们又进一步引进了高级语言。

1.1.3　高级语言

无论是机器语言还是汇编语言，它们都是面向硬件具体操作的，语言对机器的过分依赖要求使用者必须对硬件结构及其工作原理都十分熟悉，这对非计算机专业人员是难

以做到的，对于计算机的推广应用也是不利的。计算机的发展，促使人们去寻求一些与人类自然语言相接近且能为计算机所接受的语意确定、规则明确、自然直观和通用易学的计算机语言。这种与自然语言相近并为计算机所接受和执行的计算机语言称为高级语言。高级语言是面向用户的语言。无论何种机型的计算机，只要配备相应的高级语言的翻译程序，用该高级语言编写的程序就可以在该机器上运行。

例如，使用 C 语言完成例 1.1 中的计算，可用如下语句：

```
if(X<Y)
    Y=X+15;
else
    Y=X-15;
```

高级语言可读性好，机器独立性强，具有程序库，可以在运行时进行一致性检查从而检测程序中的错误，使得高级语言几乎在所有的编程领域取代了机器语言和汇编语言。高级语言也随着计算机技术的发展而不断发展，目前有许多种用于不同目的的高级程序设计语言，广泛使用的有 C、C++、Java、C#、F#、JavaScript、JSP 等。

根据人们研究兴趣的不同，高级语言也有多种不同的分类方法。从语言的范型分类，当今的大多数程序设计语言可以划分为如下四类。

1. 命令式语言

命令式语言（Imperative Language）也称过程式语言。其特点是命令驱动，面向动作（语句），即将计算看做是动作（语句）的序列。一个命令式语言程序由一系列的语句组成，每个语句的执行引起若干存储单元中的值的改变。Pascal、C 和 ADA 都是典型的命令式语言。

2. 函数式语言

函数式语言（Functional Language）注重程序实现的功能，而不是像命令式语言那样一个语句接一个语句地执行。程序的编写过程是从已有函数出发构造出更复杂的函数，对初始数据集应用这些函数直至最终的函数可以从初始数据计算出最终的结果。因此，函数式语言也称应用式语言。LISP、ML 和 Haskell 都属于这种语言。

3. 面向对象语言

面向对象语言（Object-Oriented Language）是当今最流行、最重要的语言。它的主要特点是支持封装性、继承性和多态性等。把复杂的数据和对这些数据的操作封装在一起，构成对象；对简单对象进行扩充、继承简单对象的特性，从而设计出复杂的对象。

对对象的构造可以使面向对象程序具有命令式语言的有效性，通过作用于特定数据的函数的构造，可以具有应用式语言的灵活性和可靠性。SmallTalk、C++、Java 就属于面向对象语言。

4. 逻辑式语言

逻辑式语言（Logical Language）也称做基于规则的语言（Rule-based Language）。逻辑式程序设计以“项”之间“关系”的定义、应用这些关系的事实以及从现存的事实中“推理”出新的事实的规则为基础。项可能是逻辑变量或者包含逻辑变量。事实和规则称为“子句”。逻辑式语言的程序由“子句列表”组成。最有代表性的逻辑式语言是 PROLOG。PROLOG 以逻辑程序设计为基础，以处理一阶谓词演算为背景。它文法简洁，表达力丰富，具有独特的非过程型语言（一个语句就相当于过程语言的一个子程序而并非算法的一步），是一种具有推理功能的逻辑型语言。PROLOG 语言已被广泛地应用于关系数据库、抽象问题求解、数理逻辑、公式处理、自然语言理解、专家系统以及人工智能的许多领域。

计算机的指令系统只能执行自己的指令程序，而不能执行其他语言的程序。因此，若想用高级语言，则必须有这样一种程序，它把用汇编语言或高级语言写的程序（称为源程序）翻译成等价的机器语言程序（称为目标程序），我们称这种翻译程序为翻译器。汇编语言的翻译器为汇编程序，高级语言的翻译器为编译程序。

翻译器的“翻译”通常有两种方式，即编译方式和解释方式。编译方式是：事先编好一个称为编译程序的机器语言程序，作为系统软件存放在计算机内，当用户把由高级语言编写的源程序输入计算机后，编译程序便把源程序整个地翻译成用机器语言表示的与之等价的目标程序，然后计算机再执行该目标程序，以完成源程序要处理的运算并取得结果。编译程序将源程序翻译成目标程序的过程发生在翻译时间，翻译成的目标代码随后运行的时间称为运行时间。解释方式是：源程序进入计算机时，解释程序边扫描边解释，做逐句输入逐句翻译，计算机一句句执行，并不产生目标程序。

解释器是源程序的一个执行系统，而编译程序是源程序的一个转换系统，换句话说，解释程序的工作结果是得到源程序的执行结果，因此解释程序相当于执行程序的抽象机；而编译程序的工作结果是得到等价于源程序的某种目标机程序，因此编译程序是高级语言程序到某种低级语言程序的转换器。

高级程序设计语言的编译方式和解释方式如图 1.1 所示。

C、C++、VB、VC++等高级语言执行编译方式；Java 语言则以执行解释方式为主；而 C、C++等语言是能书写编译程序的高级程序设计语言。

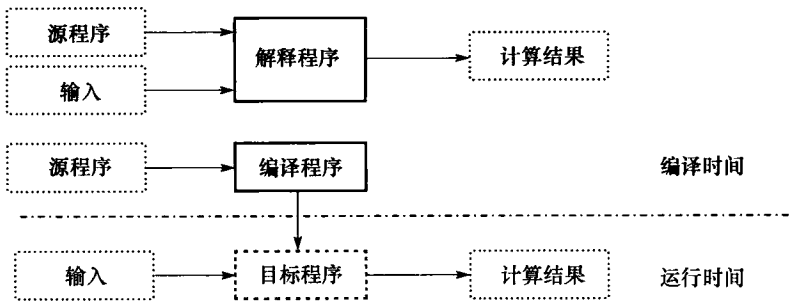


图 1.1 编译程序和解释程序

1.1.4 第四代语言

第四代语言（Fourth-Generation Language，以下简称为 4GL）的出现是出于商业需要。4GL 一词最早出现在 20 世纪 80 年代初期软件厂商的广告和产品介绍中。由于 4GL 具有“面向问题”、“非过程化程度高”等特点，可以呈数量级地提高软件生产率，缩短软件开发周期，因此赢得很多用户的青睐。20 世纪 80 年代中期，许多著名的计算机科学家对 4GL 展开了全面研究，从而使 4GL 进入了计算机科学的研究范畴。

4GL 以数据库管理系统所提供的功能为核心，进一步构造了开发高层软件系统的开发环境，如报表生成、多窗口表格设计、菜单生成系统、图形图像处理系统和决策支持系统，为用户提供了一个良好的应用开发环境。它提供了功能强大的非过程化问题定义手段，用户只需告知系统做什么，而无须说明怎么做，因此可大大提高软件生产率。

进入 20 世纪 90 年代，随着计算机软硬件技术的发展和应用水平的提高，大量基于数据库管理系统的 4GL 商品化软件已在计算机应用开发领域中获得广泛应用，成为面向数据库应用开发的主流工具，如 Oracle 应用开发环境、Informix-4GL、SQL Windows、Power Builder 等。它们为缩短软件开发周期、提高软件质量发挥了巨大的作用，为软件开发注入了新的生机和活力。

虽然 4GL 具有很多优点，也有很大的优势，成为目前应用开发的主流工具，但它也存在着以下严重不足：

- 4GL 虽然功能强大，但在其整体能力上却与 3GL 有一定的差距。这一方面是语言抽象级别提高以后不可避免地带来的（正如高级语言不能做某些汇编语言做的事情那样）；另一方面是人为带来的，许多 4GL 只面向专项应用，有的 4GL 为了提高对问题的表达能力，提供了同 3GL 的接口，以弥补其能力上的不足，如 Oracle 提供了可将 SQL 语句嵌入 C 程序中的工具 PRO＊C。
- 4GL 由于其抽象级别较高，不可避免地带来系统开销大，运行效率低（正如高级语言运行效率没有汇编语言高一样）等问题，对软硬件资源消耗严重，应用受

硬件限制。

- 由于缺乏统一的工业标准，4GL 产品花样繁多，用户界面差异很大，与具体的机器联系紧密，语言的独立性较差（SQL 稍好），影响了应用软件的移植与推广。
- 目前 4GL 主要面向基于数据库应用的领域，不适合于科学计算、高速的实时系统和系统软件开发。

1.2　程序设计语言的发展趋势

Turbo Pascal 编译器的主要编写者以及.NET 框架、Delphi 和 C#之父——Anders Hejlsberg 认为，相对于近几十年来计算机硬件的飞速发展，程序语言的改进不是很明显，主要的原因在于人们更关注“工具”、“框架”或“开发方法”的变革和创新，而忽略了语言的改进。程序设计离不开程序设计语言，程序设计语言与“工具”、“框架”或“开发方法”等一样，都对生产力有着重要影响。

Anders 认为，在过去几十年的编程历史中，程序语言的抽象级别不断提高，人们都在努力让程序语言更有表现力，这意味着人们可以用更少的代码完成更多的工作。Anders 还认为，这样的趋势还会继续保持下去，人们将看到抽象级别越来越高的语言。另外，程序语言往往倾向于构建在现有的工具上，而不会从头写起，因为每次从头开始的代价实在太高。

在 Anders 心目中，影响力较大的发展趋势主要有三种，它们分别是“声明式编程”、“动态语言”以及多核环境下的“并行编程”。此外，随着语言的发展，原本常用的“面向对象”语言、“函数式语言”或“动态语言”等边界也会变得越来越模糊，因此，“多范式”程序设计语言也是一个越发明显的发展趋势。

目前常见的编程语言都是“命令式”（Imperative）的，如 C#、Java 或 C++等。这些语言的代码更多关注的是计算任务是如何完成的（How），只要按部就班地一步步地执行写好的代码，就能实现最初的目标。所谓“声明式”（Declarative）语言的代码更多关注的是计算任务是什么（What），函数式语言就属于“声明式”语言，由于“声明式”更能凸显执行目标，也更便于进行任务分解，提高代码的并行化程度，因此，让代码包含更多的“What”，而不是“How”，是编程语言的发展趋势之一。

传统的程序设计语言都是“静态语言”，即程序的结构是固定的，程序在运行前先要经过编译，以找出程序中存在的编译错误，修改错误后再重新编译，直至没有编译错误才可以运行程序。“动态语言”是指程序在运行时可以改变其结构：引进新的函数、删除已有函数等。动态语言不区分“编译时间”（compile-time）和“运行时间”（runtime-time），动态语言的执行相对于静态语言会慢一些，也没有类型安全的概念。常

见的动态语言有 JavaScript、Python、Ruby、LISP 等。Anders 认为，静态语言和动态语言各有优势，未来的发展趋势将是二者的有机融合，而不是谁取代谁。

多核处理器以其高性能、低功耗优势正逐步取代传统的单核处理器而成为市场的主流。随着应用需求的扩大和技术的不断进步，多核必将展示出其强大的性能优势。多核处理器的出现也对程序设计语言提出新的挑战，首先是思维方式的改变，传统的并发思维，是在单 CPU 上执行多个逻辑任务，使用旧有的分时方式或时间片模型来执行多个任务。与此不同的是，多核处理器的每个 CPU 上集成了多个计算核心，多核处理器上的并发是将一个逻辑任务放在该 CPU 的多个计算核心上执行，这必将改变人们编写程序的方式，同时也意味着程序语言或者 API 要有办法来分解任务，能够把它分成多个小任务后独立执行，而传统编程语言不需要关注这些。

小结

本章介绍了程序设计语言的历史、程序设计语言的实现方式和程序设计语言的未来发展趋势。程序设计语言作为程序员与计算机交流的唯一媒介，其演变历程、实现方式和未来发展趋势，对于程序设计、程序分析、程序验证和程序的正确性证明等问题的研究有着深远的影响。程序设计语言发展至今，经历了机器语言、汇编语言、高级语言和第四代语言的演变，语言的定义和实现方法逐步完善。尤其是高级程序设计语言，根据编程机理的不同，又细化出命令式语言、函数式语言、面向对象语言、逻辑式语言四种不同范型。不同的语言范型，催生了不同的实现技术，例如编译、解释、转换等。随着应用领域的扩大和新型程序开发方法的出现，程序语言将向“多范式”方向发展。

CHAPTER 2

第 2 章　元程序设计

2.1　元程序介绍

1．元程序概念

在现代程序设计中，程序已经取代数据成为操作对象，变得越来越重要。元级程序设计系统是一种对程序进行操作的有效工具，可用于各种元程序设计。其中，对程序进行处理的基本操作称为元级操作；实现元级操作的语言叫做元语言；所处理的语言叫做目标语言。

元程序（Meta Program）是可以操作目标程序（Object Program）的程序，它可以构造目标程序，也可以将目标程序段组合成更大的目标程序，还可以观察目标程序的结构和其他特性。目标程序是以形式语言书写的一些句子，如常见的高级语言程序等。

一般来说，元程序是处理程序的程序，如编译器、解释器、类型检查器、定理证明器、程序生成器、转换系统和程序分析器等。

2．元程序和元程序设计系统的分类

从功能上看，元程序可分为两大类：程序生成器和程序分析器。程序分析器关注目标程序的结构和环境，并计算出一些值作为结果。这些结果可以是数据流图或控制流图，或者是带有原目标程序属性的另一个目标程序。该类型元级系统的例子有程序转换、优化和部分求值系统等。而程序生成器则为了解决某一类具有相似解的问题，构造另一个可解决该类问题的程序（目标程序）。通常，生成的目标程序专用于一个特定的实例，这样比使用一个一般目的的、非生成的解决方案使用的资源更少。通常，元程序可以是

程序分析器、程序生成器，或者二者的混合体。

从元语言和目标语言的异同来看，元程序设计系统有两种截然不同的种类：元语言和目标语言相同的同类系统，元语言和目标语言不同的异类系统。两种系统对描述自动的程序分析和操作都是非常有用的，但相对于异类系统而言，同类系统具有更好的教学性和实用性，因为用户只需要学习一种语言就可掌握并操作整个系统。而且它还支持反射，能够提供 n-level 程序概念——一个 n-level 目标程序自身能够成为一个操作（n+1）-level 的目标程序的元程序。然而，异类系统通常扮演更为重要的角色，带有固定的元语言元级系统，无论其构造的是每个系统都具有不同目标语言的多重系统，还是一个带有多重目标语言的单一系统，都非常具有实用价值。

3．元程序的使用

元程序设计为用户提供了很多便利。

1）性能。大多数元程序设计系统中的一个普遍的目标就是性能。元程序提供了这样一种机制，它允许用一种解释的形式书写一个多用途的程序，而不需要其他解释的辅助工具即可执行。显然，比起写一个多用途的但却低效的程序，写一个能够从一个规范生成一个高效结果的程序生成器要好很多。语法分析生成器 YACC 的使用就是一个非常鲜明的例子，我们可以从一个规范（即语言的文法）生成一个高效的分析器。

2）部分求值。部分求值是用于提高性能的另一种元级程序分析技术，通过对程序输入的部分信息的例化优化该程序，目的是在程序运行之前，识别并执行尽可能多的计算。

3）解释。元程序最普遍的用途就是从一个目标程序到另一个目标程序的解释。源语言（Source Language）和目标语言（Target Language）可以相同也可以不相同，如编译器和程序转换系统。

4）推理。元程序的另一个重要用途就是关于目标程序的推理。由于目标程序是以形式语言书写的句子，因此通过分析能够发现该程序的一些特性。利用这些特性，可以提高目标程序性能，为目标程序的行为提供保障。推理元程序的例子，如各种流分析和类型检查的程序分析。此外，推理元程序还用于构造定理证明系统，如 LEGO、HOL、Coq、Isabelle 等；逻辑框架的研究与实现，如 Elf、Twelf、LF 等。

5）移动代码。元程序已逐渐成为程序传送的一种重要手段。不同于以前使用网络传送数据给程序，目前，程序已经成为网络的传输对象。但由于安全的原因，网络传输的内容往往是程序的内在表示，为了确认传送程序的安全性和保险性，需要对其进行分析。这些被传送的程序是目标程序，对传送程序进行分析的是元程序。