TURING 图灵程序设计丛书

Apress®

Practical Common Lisp

实用Common Lisp编程

[美] Peter Seibel 著
田春 译

- 第16届Jolt生产效率奖图书
- 了解黑客青睐的编程语言
- 带你进入Common Lisp的精彩世界

人民邮电出版社
POSTS & TELECOM PRESS

“本书展示了Lisp的威力，不仅表现在传统领域上，例如仅使用短短26行代码就开发出一个完整的单元测试框架，而且还表现在一些全新的领域上，诸如解析二进制MP3文件、构建浏览歌曲集的Web应用、在Web上传播音频流等。读过本书，你将体会到Lisp具有Python等脚本语言的简洁性、C++的高效性，以及在设计语言扩展时无与伦比的灵活性。”

——Peter Norvig
Google公司研究中心主任
《人工智能：现代方法》的作者

Peter Seibel

从作家演变成程序员，又从程序员演变成作家，其职业生涯可谓一波三折。他在获得英语专业学士学位后做过一段时间的记者工作，后来被Web所吸引。在20世纪90年代早期，他用Perl建立了*Mother Jones*杂志和Organic Online网站。他作为WebLogic的早期雇员参与了Java革命，随后又在加州大学伯克利分校教授Java编程。他也是第二代Lisp程序员之一，并曾经是Symbolics的早期股东。2003年他辞去技术工作，潜心研究Lisp，并凭借本书获得Jolt生产效率大奖。2009年他出版了名噪一时的访谈录《编程人生》（*Coders at Work*）。

TURING 图灵程序设计丛书

Practical Common Lisp

实用Common Lisp编程

[美] Peter Seibel 著
田春 译

人民邮电出版社
北京

图书在版编目（CIP）数据

实用Common Lisp编程 /（美）塞贝尔（Seibel,P.）著 ; 田春译. -- 北京 : 人民邮电出版社, 2011.10
（图灵程序设计丛书）
书名原文: Practical Common Lisp
ISBN 978-7-115-26374-2

Ⅰ. ①实… Ⅱ. ①塞… ②田… Ⅲ. ①LISP语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2011)第191310号

内 容 提 要

这是一本不同寻常的 Common Lisp 入门书。本书首先从作者的学习经过及语言历史出发，随后用 21 个章节讲述了各种基础知识，主要包括：REPL 及 Common Lisp 的各种实现、S- 表达式、函数与变量、标准宏与自定义宏、数字与字符以及字符串、集合与向量、列表处理、文件与文件 I/O 处理、类、FORMAT 格式、符号与包，等等。而接下来的 9 个章节则翔实地介绍了几个有代表性的实例，其中包含如何构建垃圾过滤器、解析二进制文件、构建 ID3 解析器，以及如何编写一个完整的 MP3 Web 应用程序等内容。最后还对一些未介绍内容加以延伸。

本书内容适合 Common Lisp 初学者及对之感兴趣的相关人士。

图灵程序设计丛书

实用Common Lisp编程

◆ 著 [美] Peter Seibel
译 田 春
责任编辑 傅志红

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
北京 印刷

◆ 开本：800×1000 1/16
印张：27.25
字数：678千字 2011年 10 月第 1 版
印数：1－3 500册 2011年 10 月北京第 1 次印刷

著作权合同登记号 图字：01-2010-2360 号

ISBN 978-7-115-26374-2

定价：89.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

目　录

第 1 章 绪言：为什么是Lisp

如果你认为编程最大的乐趣在于，可以用简明扼要的代码来清晰表达你的意图，完成许多事，那么使用Common Lisp编程可以说是用计算机所能做到的最有趣的事了。比起相当多的其他计算机语言，它可以让你更快地完成更多工作。

这是个大胆的断言。可我要怎样证明呢？当然不是用本章的只言片语了，而是这一整本书。你必须先学习一些Lisp知识才能体会到这一点。不过眼下，让我们先介绍一些轶事，即我本人迈向Lisp编程之路的经历。然后在下一节里，我将说明你可以通过学习Common Lisp得到些什么收获。

我是极少数的所谓第二代Lisp黑客之一。我父亲的计算机生涯开始于用汇编语言给他的机器编写操作系统，从而为他的物理学博士论文搜集资料。在成功帮助了几家物理实验室用上计算机系统之后，从20世纪80年代起，他彻底抛弃了物理学，转而为一家大型制药公司工作。当时那家公司里有个软件项目，正在开发用于模拟化工生产过程的软件，比如说，增大容器的尺寸将会怎样影响其年产量。原先的团队使用FORTRAN进行开发，已经耗费了该项目的一半预算和几乎全部的时间，却没能交付任何成果。那是在20世纪80年代，人工智能（AI）蓬勃发展，Lisp正在流行。于是，我父亲（当时还不是Lisp程序员）来到卡内基－梅隆大学（CMU），向那些正在开发后来被称为Common Lisp语言的人们咨询：如果用Lisp来开发这个项目是否合适？

CMU的人向我父亲演示了一些他们正在做的东西，于是他被说服了。然后他又说服老板让他的团队接手那个失败的项目并用Lisp重新开发。一年以后，仅仅使用了原先剩余的预算，他的团队就交付了一个可用的应用程序，而且还带有已经被原先团队所放弃的一些功能。我父亲将其团队成功的原因归结为他们使用了Lisp。

当然，仅此一例还不足以说明问题，而且也有可能我父亲对其成功原因认识有误，或者也许Lisp仅仅在那个年代才可以跟其他语言相媲美。如今我们有了大量精巧的新语言，它们中的许多都吸收了Lisp的特性。是否真的可以说，Lisp在现今也具有跟我父亲那个年代一样的优势吗？请继续往下读。

尽管父亲作了大量努力，但我在高中前没有学过一点儿Lisp。在大学时代我也没怎么用任何语言来写程序，是后来出现的Web让我回到了计算机的怀抱。我首先使用的是Perl，先是为*Mother Jones*杂志的网站构建在线论坛，待经验丰富以后转向Web商店Organic Online，在那里我为当时的大型Web站点工作，例如耐克公司在1996年奥运会期间上线的站点。后来我转向Java，并成为

WebLogic（现在是BEA的一部分）的早期开发者。离开WebLogic以后，我又作为另一个团队的首席程序员开始用Java编写事务消息系统。在此期间，对编程语言的广泛兴趣使我充分研究了诸如C、C++和Python这些主流语言，以及Smalltalk、Eiffel和Beta这些小众语言。

所以我对Perl和Java两种语言了如指掌，同时还熟悉其他不下六种语言。不过最终我还是意识到，我对编程语言的兴趣其实来源于记忆之中的我父亲的Lisp经历——语言之间真的有天壤之别，尽管所有的编程语言在形式上都是图灵等价的，但一些语言真的会比其他语言更快更好地完成某些事情，并且在使用的过程中还能给你带来更多的乐趣。不过可笑的是，我从没有在Lisp上花太多时间。于是我开始利用业余时间研究Lisp。无论我做什么，最令我兴奋的始终是可以很快地将思路变成可用的代码。

举个例子，在一次假期里，我差不多在Lisp上花了一个星期的时间，目标是实现一个基于遗传算法、用来下围棋的程序系统——我以前做Java程序员的时候已经写过了。虽然我那时的Common Lisp知识极其有限，还要不时地查询基本函数的用法，但我还是能感觉到比用Java重写同样的程序来得顺手——尽管自编写该程序的最初版本以来，我又多了几年的Java经验。

类似的经历还产生了我将在第24章里讨论的一个库。我以前在WebLogic的时候曾经用Java写了一个库，用来解析Java的类文件。虽然可以正常使用，但是代码有点乱，并且难以修改或扩展。几年来我曾多次重写这个库，并期望凭借日益提高的Java技巧可以找到某种方式，来消除其中大量的重复代码，可惜从未成功。但当我改用Common Lisp做这件事时，我只花了两天时间，最后不但得到了一个Java类文件的解析器，而且还产生一个可用于解析任何二进制文件的通用库。第24章将讨论这个库的工作原理，第25章会用它写一个MP3文件的内嵌ID3标签的解析器。

1.1　为什么是 Lisp

很难用绪言这一章的寥寥数页来说清楚为何一门语言的用户会喜欢这门语言，更难的是找出一个理由说服你花时间来学习某种语言。现身说法只能到此为止了。也许我喜欢Lisp是因为它刚好符合我的思维方式，甚至可能是遗传因素，因为我父亲也喜欢它。因此，在你开始学习Lisp之前，想要知道能得到什么回报也是合理的。

对某些语言来说，回报是相对明显的。举个例子，如果打算编写Unix上的底层代码，那你应该去学C。或者如果打算编写特定的跨平台应用程序，那你应该去学Java。而相当数量的公司仍在大量使用C++，如果你打算在这些公司里找到工作，那就应该去学C++。

尽管如此，对于多数语言来说，回报往往不是那么容易界定的，这里面存在包括个人感情在内的主观因素。Perl拥趸们经常说Perl“让简单的事情简单，让复杂的事情可行”，并且沉迷于“做事情永远都有不止一种方法”[①]这一Perl格言所推崇的事实。另一方面，Python爱好者们认为Python是简洁的，并且Python代码之所以易于理解，是因为正如他们的格言所说：“做事情只有一种方法。”

那么Common Lisp呢？没有迹象表明采用Common Lisp可以立即得到诸如C、Java和C++那样显而易见的好处（当然了，除非刚好拥有一台Lisp机）。使用Lisp所获得的好处在很大程度上取决

① Perl作为“因特网的传送带/血管”也同样值得学习。

于使用经验。本书的其余部分将向读者展示Common Lisp的特性以及如何使用它们，让你自己去感受它。眼下，我只想让你先对Lisp哲学有个大致的感受。

Common Lisp中最接近格言的是一句类似禅语的描述："可编程的编程语言。"虽然隐晦了一些，但这句话却道出了Common Lisp至今仍然雄踞其他语言之上的最大优势。Common Lisp比其他任何语言都更加遵循一种哲学——凡利于语言设计者的也利于语言使用者。这就意味着，当使用Common Lisp编程的时候，你永远不会遇到这种情况：语言里刚好缺乏某些可能令程序更容易编写的特性，因为正如你将在本书中看到的，你可以为语言添加任何想要的特性。

因此，Common Lisp程序倾向于把关于程序如何工作的想法更清楚地映射到实际所写的代码上。想法永远不会被过于紧凑的代码和不断重复的模式搞得含糊不清。这将使代码更加易于维护，因为不必在每次修改之前都先复查大量的相关代码。甚至，对程序行为的系统化调整也经常可以通过对实际代码作相对少量的修改来实现。这也意味着可以更快速地开发，编写更少的代码，也不必再花时间在语言的限制里寻求更简洁的方式来表达想法了。[①]

Common Lisp也是一门适合做探索性编程的优秀语言。如果在刚开始编写程序的时候对整个工作机制还不甚明了，Common Lisp提供了一些特性可以有助于实现递进的交互式开发。

比如，将在下一章介绍的交互式"读–求值–打印"循环，它可以让你在开发过程中持续地与程序交互。编写新函数、测试它、修改它、尝试不同的实现方法，从而使思路不会因漫长的编译周期而停滞下来。[②]

其他支持连贯的交互式编程风格的语言特性，还包括Lisp的动态类型机制以及Common Lisp状况系统（condition system）。由于前者的存在，只需花较少的时间就能让编译器把代码跑起来，然后把更多的时间放在如何实际运行和改进代码上[③]，而后者甚至可以实现交互式地开发错误处理代码。

---

① 遗憾的是，在不同语言的生产力方面几乎没有实际的研究。一份阐明了在程序员和程序效率的组合上，Lisp相比于C++和Java脱颖而出的报告在http://www.norvig.com/java-lisp.html上有所讨论。

② 心理学家已经鉴别出大脑的一种称为连贯性（flow）的状态，这时我们可以产生高度的注意力和生产力。连贯性对于编程的重要性在近二十年以前就已经被认识到了，其最早在一本经典的关于编程的人为因素的书《人件：富有成果的项目和团队》（*Peopleware: Productive Projects and Teams*，Tom Demarco和Timothy Lister著，Dorset Hourse，1987年）里讨论过。连贯性的两个关键因素是人们需要15分钟才能进入连贯性状态，而即便短暂的打岔也会使人完全退出这一状态，从而需要另外15分钟才能重新进入状态。DeMarco和Lister，以及多数后来的作者，都很反感诸如电话铃和老板的贸然来访这类破坏连贯性的打岔。较少被考虑到但可能对程序员同样重要的是我们工具所造成的打岔，例如，那些在你测试最新代码之前需要经历冗长的编译过程的语言，其对于连贯性的影响可能跟吵闹的电话铃或者爱管闲事的老板同样有害。因此，Lisp作为一种可以令你保持连贯状态的语言，也是你应该了解它的一个理由。

③ 至少对某些人来说，这个观点很可能有争议。静态和动态类型的信仰之争在编程领域由来已久。如果你信奉C++和Java（或者是诸如Haskell和ML的静态类型函数式编程语言），并且拒绝生活在没有静态类型检查的环境里，你可能会就此合上本书了。不过，在此之前，你最好先查查"静态类型偏执狂"Robert Martin[*Design Object Oriented C++ Applications Using the Booch Method*(Prentice Hall, 1995年)的作者]以及C++和Java领域的作者Bruce Eckel[*Thinking in C++*（Prentice Hall, 1995年）和*Thinking in Java*(Prentice Hall, 1998年)的作者]在他们的博客（http://www.artima.com/weblogs/viewpost.jsp?thread=4639和http://www.mindview.net/WebLog/log-0025）上是怎样自我描述的。另一方面，信奉SmallTalk、Python、Perl或者Ruby的人们应该会对Common Lisp的这方面感觉良好。

作为一门“可编程的编程语言”，Common Lisp除了支持各种小修小补以便开发人员更容易地编写某些程序之外，对于那些从根本上改变编程方式的新思想的支持也是绰绰有余的。例如，Common Lisp强大的对象系统CLOS（Common Lisp Object System），其最初的实现就是一个用可移植的Common Lisp写成的库。而在这个库正式成为语言的一部分之前，Lisp程序员就可以体验其实际功能了。

目前来看，无论下一个流行的编程范例是什么，Common Lisp都极有可能在无需修改其语言核心部分的情况下将其吸纳进来。例如，最近有个Lisp程序员写了一个叫做AspectL的库，它为Common Lisp增加了对面向方面编程（AOP）的支持。[①]如果AOP将主宰编程的未来，那么Common Lisp将可以直接支持它而无需对基础语言作任何修改，并且也不需要额外的预处理器和编译器。[②]

1.2　Lisp 的诞生

Common Lisp是1956年John McCarthy发明的Lisp语言的现代版本。Lisp在1956年被设计用于“符号数据处理”[③]，而Lisp这个名字本身就来源于其最擅长的工作：列表处理（LISt Processing）。从那时起，Lisp得到了长足的发展。Common Lisp引人瞩目地具备一系列现代数据类型：将在第19章里介绍的状况系统提供了Java、Python和C++等语言的异常系统里所没有的充分灵活性，强大的面向对象编程支持，以及其他编程语言里完全不存在的一些语言机制。这一切怎么可能呢？怎么会进化出如此装备精良的语言来呢？

原来，McCarthy曾经是（现在也是）一名人工智能（AI）研究者，他在Lisp语言的最初版本里内置的很多特性，使其成为了AI编程的绝佳语言。在AI繁荣昌盛的20世纪80年代，Lisp始终是程序员们所偏爱的工具，广泛用于编写软件来求解包括自动定理证明、规划和调度以及计算机视觉在内的各种难题。这些问题都需要大量难于编写的软件，为此，AI程序员们需要一门强大的语言，而他们就将Lisp发展成了这样一门语言。另外，冷战也起了积极的作用——五角大楼向国防高级研究规划局（DARPA）投入了大量资金，其中的相当一部分用于研究诸如大规模战场模拟、自动规划以及自然语言接口等问题。这些研究人员也在使用Lisp并且持续地对其进行改进以满足自身需要。

推动Lisp特性进化的动力也同样推动了其他相关领域的发展——大型的AI问题无论如何编码总是要耗费大量的计算资源，而如果按照摩尔定律倒推20年，你就可以想象20世纪80年代的计

---

① AspectL是一个跟它的Java版前任AspectJ同样有趣的项目，后者由Common Lisp对象和元对象系统的设计者之一Gregor Kiczales所作。对许多Lisp程序员来说，AspectJ看起来就像是Kiczales尝试将其思想从Common Lisp向后移植到了Java。尽管如此，AspectL的作者Pascal Costanza认为，AOP许多有趣的思想可能对Common Lisp有用。当然，他能够将AspectL以库的形式实现的根本原因在于，Kiczales所设计的Common Lisp元对象协议（Meta Object Protocol）具有难以想象的灵活性。为了实现AspectJ，Kiczales不得不写了一个单独的编译器，将一种新语言编译成Java源代码。AspectL项目的主页是http://common-lisp.net/project/aspectl/。

② 或者我们可以从另一个技术上更精确的方面来看待这件事：Common Lisp提供了内置的功能以方便集成嵌入式语言的编译器。

③ *Lisp 1.5 Programmer's Manual*（麻省理工学院出版社, 1962年）。

算资源是何等的贫乏了。Lisp工作者们不得不想尽办法从实现中获得更多的性能。现代Common Lisp实现就是这些早期工作的结晶，它们通常都带有相当专业的可产生原生机器码的编译器。感谢摩尔定律，今天我们从任何纯解释型语言里也能获得可接受的性能了，性能对于Common Lisp来说再也不是问题了。不过，你在第32章可以看到，通过使用适当的（可选）变量声明，一个好的Lisp编译器所生成的机器码，完全可以跟C编译器生成的机器码相媲美。

20世纪80年代也是Lisp机的年代，当时好几家公司（其中最著名的是Symbolics）都在生产可以在芯片上直接运行Lisp的计算机系统。Lisp因此成了系统编程语言，被广泛用于编写操作系统、编辑器、编译器，以及Lisp机上的大量其他软件。

事实上，到了20世纪80年代早期，几家AI实验室和Lisp机厂商都提供了他们自己的Lisp实现，众多的Lisp系统和方言让DARPA开始担心Lisp社区可能走向分裂。为了应对这些担忧，一个由Lisp黑客组成的草根组织于1981年成立，旨在结合既有Lisp方言之所长，定义一种新的称为Common Lisp的标准化Lisp语言。最后，他们的工作成果记录在了Guy Steele的*Common Lisp the Language*（*CLtL*，Digital press，1984年）一书里，该书相当于Lisp的圣经。

到1986年的时候，首批Common Lisp实现诞生了，它们是在Common Lisp试图取代的那些方言的基础上写成的。1996年，美国国家标准学会（ANSI）发布了一个建立在CLtL之上并加以扩展的Common Lisp标准，其中增加了一些主要的新特性，包括CLOS和状况系统。但事情还没结束：跟此前的CLtL一样，ANSI标准有意为语言实现者保留一定 的空间，以试验各种最佳的工作方式。一个完整的Lisp实现将带有丰富的运行时环境，并提供GUI微件、多线程控制和TCP/IP 套接字等。今天的Common Lisp则进化得更像其他的开源语言——用户可以编写他们所需要的库并开放给其他人使用。在过去的几年里，开源Lisp库领域尤为活跃。

所以，一方面，Lisp是计算机科学领域的“经典语言”之一，构建在经过时间考验的各种思想之上。[①]另一方面，它完全是一门现代的通用语言，其设计反映了尽可能高效可靠地求解实际问题的实用主义观点。Lisp“经典”遗产的唯一缺点是，许多人仍然生活在片面的Lisp背景之下，他们可能只是在McCarthy发明Lisp以来的近半个世纪中的某些特定时刻接触到了这门语言的某些方面。如果有人告诉你Lisp只能被解释执行，因此会很慢，或者你不得不用递归来干每件事，那么一定要问问他们究竟在谈论哪种Lisp方言，以及他们是否是在计算机远古时代学到这些东西的。[②]

---

① 首先由Lisp引进的编程思想包括if/then/else控制结构、递归函数调用、动态内存分配、垃圾收集、第一类（first-class）函数、词法闭包、交互式编程、增量编译以及动态类型。

② 关于Lisp的一个最常见的说法是该语言“已死”。虽然Common Lisp确实不如Visual Basic或者Java这些语言使用广泛，但把一个继续用于新的开发并且继续吸引新用户的语言描述成“已死”看起来有些奇怪。一些近期的Lisp成功案例包括Paul Graham的Viaweb，在Yahoo买下了Paul的公司以后变成了Yahoo Store；由在线机票销售商Orbitz等使用的ITA Software的航空票务系统QPX；Naughty Dog运行在PlayStation 2上的游戏Jak and Daxter，在很大程度上是用Naughty Dog发明的一种称为GOAL的特定领域Lisp方言写成的，其编译器本身是用Common Lisp写的；还有自动机器人真空吸尘器Roomba，其软件是用L语言写的，后者是Common Lisp的向下兼容子集。也许最有说服力的是承载开源Common Lisp项目的Web站点Common-Lisp.net的成长，以及各种本地Lisp用户组过去几年里在数量上的显著增长。

我曾经学过Lisp，不过跟你所描述的不太一样

如果你以前用过Lisp，你对Lisp的认识很可能对学习Common Lisp没什么帮助。尽管Common Lisp取代了大多数它所继承下来的方言，但它并非仅存的Lisp方言。你要清楚自己是在何时何地认识Lisp的，很有可能你学的是某种其他方言。

除了Common Lisp以外，另一种仍然有着活跃用户群的通用Lisp方言是Scheme。Common Lisp从Scheme那里吸收了一些重要的特性，但从未试图取代它。

Scheme最早在MIT设计出来，然后很快用作本科计算机科学课程的教学语言，它一直被定位成一种与Common Lisp有所不同的语言。特别是Scheme的设计者们将注意力集中在使其语言核心尽可能地小而简单。这对作为教学语言来说非常有用，编程语言研究者们也很容易形式化地证明语言本身的有关命题。

这样设计的另一个好处是使得通过标准规范理解整个语言变得相对简单，但这样做带来的问题是缺失了许多在Common Lisp里已经标准化了的有用特性。个别的Scheme实现者可能以特定的实现方式提供了这些特性，但它们在标准中的缺失则使得编写可移植的Scheme代码比编写可移植的Common Lisp代码更加困难。

Scheme同样强调函数式的编程风格，并且使用了比Common Lisp更多的递归。如果在大学里学过Lisp并且感觉它只是一种没有现实应用的学术语言，那你八成是学了Scheme。当然，说Scheme具有这样的特征并不是很公正，只不过同样的说法用在Common Lisp身上更加不合适罢了，后者专门被设计成真实世界的工程语言，而不是一种理论上的“纯”语言。

如果学过Scheme，也应该当心，Scheme和Common Lisp之间的许多细微差别可能会使人犯错。这些差别也是Common Lisp和Scheme社区的狂热分子之间一些长期信仰之争的导火索。日后随着讨论的深入，我将指出其中最重要的差别。

另外两种仍然广泛使用的Lisp方言是Elisp和Autolisp。Elisp是Emacs编辑器的扩展语言，而Autolisp是Autodesk的AutoCAD计算机辅助设计工具的扩展语言。尽管用Elisp和Autolisp可能已经写出了超过任何其他方言的代码行数，但它们都不能用在各自的宿主应用程序之外，而且它们无论与Scheme还是Common Lisp相比都是相当过时的Lisp方言了。如果你曾经用过这些方言的一种，就需要做好准备，你可能要在Lisp时间机器里向前跳跃几十年了。

1.3　本书面向的读者

如果你对Common Lisp感兴趣，那么无论是否已经确定要使用它或者只是想一窥其门径，本书都挺适合你的。

如果你已经学会了一些Lisp，但却难于跨越从学术训练到真实程序之间的鸿沟，那么本书刚好可以帮你走上正途。而另一方面，你也不必带着学以致用的目的来阅读本书。

如果你是个顽强的实用主义者，想知道Common Lisp相比Perl、Python、Java、C和C#这些语言都有哪些优势，那么本书应该可以提供一些思路。或者你根本就没打算使用Lisp——可能是因为已经确信Lisp并不比已知的其他语言更好，但由于不熟悉它而无法反驳那些Lisp程序员。如果

是这样，本书将给你一个直奔Common Lisp主题的介绍。如果在读完本书以后，你仍然认为Common Lisp赶不上自己当前喜爱的其他语言，那么你将有充分理由来说明自己的观点了。

我不但介绍了该语言的语法和语义，还讲述了如何使用它来编写有用的软件。在本书的第一部分，我将谈及语言本身，同时穿插一些实践性章节，展示如何编写真实的代码。接着，在我阐述了该语言的绝大部分内容后——包括某些在其他书里往往留给你自己去考查的内容，将给出九个更实用的章节，帮助你编写一些中等大小的程序来做一些你可能认为有用的事：过滤垃圾邮件、解析二进制文件、分类MP3、通过网络播放MP3流媒体，以及为MP3目录和服务器提供Web接口。

读完本书后，你将熟悉该语言的所有最重要的特性以及它们是如何组织在一起的，而且你已经用Common Lisp写出了一些非凡的程序，并且可以凭借自身力量继续探索该语言了。尽管每个人的Lisp之路都有所不同，但我还是希望本书可以帮助你少走些弯路。那么，让我们就此上路吧。

田春

网名“冰河”，Glority Software资深软件工程师，前网易杭州研究院高级开发工程师和系统管理员，资深Common Lisp程序员。2001～2005年就读于浙江大学。2003年起开始学习Commom Lisp，精通Lisp史和各种实现，2007年起成为 LispWorks 付费用户，Common Lisp社区的网络专家，开源项目cl–net–snmp（SNMP协议库）的作者，usocket跨平台网络库的主要维护者，common–lisp.net站点管理员，水木社区（newsmth.net）函数型编程语言（FuncProgram）版主，美国Versata/Gensym公司技术顾问。曾在2008年翻译了Paul Graham的*On Lisp*一书，在ILC 2009（国际Lisp会议）上发表学术论文，在《程序员》杂志上发表Common Lisp专题文章，并在网上撰写过大量相关的技术文章。

Practical Common Lisp

实用Common Lisp编程

Lisp是计算机科学领域的经典语言之一，它被许多资深程序员视为编程语言中的圣杯。Lisp语言由约翰·麦卡锡提出，它构建在久经考验的各种编程思想之上，其设计反映了尽可能高效可靠地求解实际问题的实用主义观点。

《Unix编程艺术》的作者Eric Raymond认为Lisp语言对黑客特别重要，他在“如何成为黑客”一文中说过：“Lisp很值得学习。掌握它以后，你会感受到它带来的极大启发。这会大大提高你的编程水平，使你成为一名更好的程序员，即使你在实际工作中很少用到Lisp。”

《黑客与画家》的作者Paul Graham更是Lisp的倡导者，他在宣传Lisp的文章“拒绝平庸”中说到：“Lisp语言的好处不在于它有一些狂热爱好者才明白的优点，而在于它是目前最强大的编程语言。它没有得到广泛使用的原因就是因为编程语言不仅仅是技术，也是一种习惯性思维，而习惯非常难以改变。”

书中展示了如何用Common Lisp编写一些真实可用的软件，远远超越了笨拙的学院派训练或简单的编辑器定制，读者可以发现，Lisp即便在其许多特性已被其他语言采纳以后仍然有其独到之处。书中用超过三分之一的篇幅讲述开发具有一定复杂度的软件——基于统计的垃圾过滤器、用来解析二进制文件的库，以及一个带有完整在线MP3数据库和Web接口、通过网络流传输MP3的服务器。

Apress®

图灵社区：www.ituring.com.cn
反馈/投稿/推荐信箱：contact@turingbook.com
热线：(010)51095186转604

分类建议 计算机/程序设计

人民邮电出版社网址：www.ptpress.com.cn

ISBN 978-7-115-26374-2
定价：89.00元