

<<编译原理>>

图书基本信息

书名：<<编译原理>>

13位ISBN编号：9787111251217

10位ISBN编号：7111251210

出版时间：2008年12月

出版时间：机械工业出版社

作者：Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman

页数：631

译者：赵建华, 郑滔, 戴新宇

版权说明：本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问：<http://www.tushu007.com>

<<编译原理>>

前言

绝大部分软件是使用高级程序设计语言来编写的。

用这些语言编写的软件必须经过编译器的编译，才能转换为可以在计算机上运行的机器代码。

编译器所生成代码的正确性和质量会直接影响成千上万个软件。

因此，编译器构造原理和技术是计算机科学技术领域中的一个非常重要的组成部分。

不仅如此，编译技术在当前已经广泛应用于编译器构造之外的其他领域，比如程序分析 / 验证、模型转换、语言处理等领域。

因此，虽然大部分读者不会参与设计商用编译器，但拥有编译的相关知识仍然会对他们的研究开发生涯产生有益的影响。

A . V . Aho等人撰写的《Compilers : Principles , Techniques , and Yools》被誉为编译教科书中的“龙书”。

这说明这本书具有很高的权威性。

我们有幸受机械工业出版社的委托，翻译龙书的第2版。

本书不仅包含了词法分析、语法分析、语义分析、代码生成等传统、经典的编译知识，还详细介绍了一些最新研究成果，比如过程间指针分析的最新进展。

这使得本书不仅适用于编译原理的初学者，还可以作为研究人员的参考书目。

本书不仅介绍编译器构造的基本原理和技术，还详细介绍一些有用的编译器构造工具，比如对Lex和Yacc的介绍使得读者可以了解这些工具的工作原理和使用方法。

除此之外，读者还可以看到很多其他领域的概念在编译器构造中的应用。

比如在第9章，读者可以看到群论中的抽象概念“格”被完美地应用于数据流分析算法的设计。

而在第11章，线性规划和整数规划技术被成功地应用于程序并行化技术。

这些内容对拓展读者的视野和思路有很大的好处。

由于本书覆盖的范围非常广，不可能在一个学期内讲完本书的全部内容。

因此我建议采用本书作为本科生教材的老师只选择讲授其中的基础部分，即第1章到第9章中的大部分内容。

第2章是对后面各章内容的介绍，可以在讲授相应内容之前指导学生预习。

最后感谢机械工业出版社的温莉芳女士以及姚蕾和朱劫两位编辑在本书的翻译过程中给予我们的有力帮助，也感谢其他给予我们支持的同事。

由于水平有限，翻译中的错漏之处在所难免，欢迎读者批评指正。

<<编译原理>>

内容概要

本书全面、深入地探讨了编译器设计方面的重要主题，包括词法分析、语法分析、语法制导定义和语法制导翻译、运行时刻环境、目标代码生成、代码优化技术、并行性检测以及过程间分析技术，并在相关章节中给出大量的实例。

与上一版相比，本书进行了全面修订，涵盖了编译器开发方面最新进展。

每章中都提供了大量的实例及参考文献。

本书是编译原理课程方面的经典教材，内容丰富，适合作为高等院校计算机及相关专业本科生及研究生的编译原理课程的教材，也是广大技术人员的极佳参考读物。

<<编译原理>>

作者简介

Alfred

V. Aho, 美国哥伦比亚大学教授, 美国国家工程院院士, ACM和IEEE会士, 曾获得IEEE的冯·诺伊曼奖。

著有多部算法、数据结构、编译器、数据库系统及计算机科学基础方面的著作。

<<编译原理>>

书籍目录

出版者的话

译者序

前言

第1章 引论

1.1 语言处理器

1.2 一个编译器的结构

1.2.1 词法分析

1.2.2 语法分析

1.2.3 语义分析

1.2.4 中间代码生成

1.2.5 代码优化

1.2.6 代码生成

1.2.7 符号表管理

1.2.8 将多个步骤组合成趟

1.2.9 编译器构造工具

1.3 程序设计语言的发展历程

1.3.1 走向高级程序设计语言

1.3.2 对编译器的影响

1.3.3 1.3节的练习

1.4 构建一个编译器的相关科学

1.4.1 编译器设计和实现中的建模

1.4.2 代码优化的科学

1.5 编译技术的应用

1.5.1 高级程序设计语言的实现

1.5.2 针对计算机体系结构的优化

1.5.3 新计算机体系结构的设计

1.5.4 程序翻译

1.5.5 软件生产率工具

1.6 程序设计语言基础

1.6.1 静态和动态的区别

1.6.2 环境与状态

1.6.3 静态作用域和块结构

1.6.4 显式访问控制

1.6.5 动态作用域

1.6.6 参数传递机制

1.6.7 别名

1.6.8 1.6节的练习

1.7 第1章的总结

1.8 第1章的参考书目

第2章 一个简单的语法制导翻译器

2.1 引言

2.2 语法定义

2.2.1 文法定义

2.2.2 推导

2.2.3 语法分析树

<<编译原理>>

- 2.2.4 二义性
 - 2.2.5 运算符的结合性
 - 2.2.6 运算符的优先级
 - 2.2.7 2.2节的练习
 - 2.3 语法制导翻译
 - 2.3.1 后缀表示
 - 2.3.2 综合属性
 - 2.3.3 简单语法制导定义
 - 2.3.4 树的遍历
 - 2.3.5 翻译方案
 - 2.3.6 2.3节的练习
 - 2.4 语法分析
 - 2.4.1 自顶向下分析方法
 - 2.4.2 预测分析法
 - 2.4.3 何时使用产生式
 - 2.4.4 设计一个预测语法分析器
 - 2.4.5 左递归
 - 2.4.6 2.4节的练习
 - 2.5 简单表达式的翻译器
 - 2.5.1 抽象语法和具体语法
 - 2.5.2 调整翻译方案
 - 2.5.3 非终结符号的过程
 - 2.5.4 翻译器的简化
 - 2.5.5 完整的程序
 - 2.6 词法分析
 - 2.6.1 剔除空白和注释
 - 2.6.2 预读
 - 2.6.3 常量
 - 2.6.4 识别关键字和标识符
 - 2.6.5 词法分析器
 - 2.6.6 2.6节的练习
 - 2.7 符号表
 - 2.7.1 为每个作用域设置一个符号表
 - 2.7.2 符号表的使用
 - 2.8 中间代码生成
 - 2.8.1 两种中间表示形式
 - 2.8.2 语法树的构造
 - 2.8.4 三地址码
 - 2.8.5 2.8节的练习
 - 2.9 第2章的总结
- 第3章 词法分析
- 3.1 词法分析器的作用
 - 3.1.1 词法分析及解析
 - 3.1.2 词法单元、模式、词素
 - 3.1.3 词法单元的属性
 - 3.1.4 词法错误
 - 3.1.5 3.1节的练习

<<编译原理>>

- 3.2 输入缓冲
 - 3.2.1 缓冲区间
 - 3.2.2 哨兵标记
- 3.3 词法单元的规约
 - 3.3.1 串和语言
 - 3.3.2 语言上的运算
 - 3.3.3 正则表达式
 - 3.3.4 正则定义
 - 3.3.5 正则表达式的扩展
 - 3.3.6 3.3节的练习
- 3.4 词法单元的识别
 - 3.4.1 状态转换图
 - 3.4.2 保留字和标识符的识别
 - 3.4.3 完成我们的连续性例子
 - 3.4.4 基于状态转换图的词法分析器的体系结构
 - 3.4.5 3.4节的练习
- 3.5 词法分析器生成工具Lex
 - 3.5.1 Lex的使用
 - 3.5.2 Lex程序的结构
 - 3.5.3 Lex中的冲突解决
 - 3.5.4 向前看运算符
 - 3.5.5 3.5节练习
- 3.6 有穷自动机
 - 3.6.1 不确定的有穷自动机
 - 3.6.2 转换表
 - 3.6.3 NFA接受输入字符串
 - 3.6.4 确定的有穷自动机
 - 3.6.5 3.6节的练习
- 3.7 从正则表达式到自动机
 - 3.7.1 从NFA到DFA的转换
 - 3.7.2 NFA的模拟
 - 3.7.3 NFA模拟效率
 - 3.7.4 从正则表达式构造NFA
 - 3.7.5 字符串处理算法的效率
 - 3.7.6 3.7节的练习
- 3.8 词法分析器生成工具的设计
 - 3.8.1 被生成的词法分析器的结构
 - 3.8.2 基于NFA的模式匹配
 - 3.8.3 词法分析器使用的DFA
 - 3.8.4 实现向前看运算符
 - 3.8.5 3.8的练习
- 3.9 基于DFA的模式匹配器的优化
 - 3.9.1 NFA的重要状态
 - 3.9.2 根据抽象语法树计算得到的函数
 - 3.9.3 计算nullable、firstpos及lastpos
 - 3.9.4 计算followpos
 - 3.9.5 根据正则表达式构建DFA

<<编译原理>>

- 3.9.6 最小化一个DFA的状态数
- 3.9.7 词法分析器的状态最小化
- 3.9.8 在DFA模拟中用时间换取空间
- 3.9.9 3.9节的练习
- 3.9.10 第3章的总结
- 3.11 第3章参考文献
- 第4章 语法分析
 - 4.1 引论
 - 4.1.1 语法分析器的角色
 - 4.1.2 代表性的文法
 - 4.1.3 语法错误的处理
 - 4.1.4 错误恢复策略
 - 4.2 上下文无关文法
 - 4.2.1 上下文无关文法的正式定义
 - 4.2.2 符号表示的惯例
 - 4.2.3 推导
 - 4.2.4 语法分析树和推导
 - 4.2.5 二义性
 - 4.2.6 验证文法生成的语言
 - 4.2.7 上下文无关文法和正则表达式
 - 4.2.8 4.2节的练习
 - 4.3 设计文法
 - 4.3.1 词法分析和语法分析
 - 4.3.2 消除二义性
 - 4.3.3 左递归的消除
 - 4.3.4 提取左公因子
 - 4.3.5 非上下文无关的语言构造
 - 4.3.6 4.3节的练习
 - 4.4 自顶向下的语法分析
 - 4.4.1 递归下降的语法分析
 - 4.4.2 FIRST和FOLLOW
 - 4.4.3 LL(1)文法
 - 4.4.4 非递归的预测分析
 - 4.4.5 预测分析中的错误恢复
 - 4.4.6 4.4节的练习
 - 4.5 自底向上的语法分析
 - 4.5.1 归约
 - 4.5.2 句柄剪枝
 - 4.5.3 移入-归约语法分析技术
 - 4.5.4 移入-归约语法分析中的冲突
 - 4.5.5 4.5节的练习
 - 4.6 LR语法分析技术介绍：简单LR技术
 - 4.6.1 为什么使用LR语法分析器？
 - 4.6.2 项和LR(0)自动机
 - 4.6.3 LR-语法分析算法
 - 4.6.4 构造SLR-分析表

<<编译原理>>

- 4.6.5 可行前缀
- 4.6.6 4.6节的练习
- 4.7 更强大的LR语法分析器
 - 4.7.1 规范LR(1)项
 - 4.7.2 构造LR(1)项集
 - 4.7.3 规范LR(1)分析表
 - 4.7.4 构造LALR语法分析表
 - 4.7.5 LALR语法分析表的高效构造方法
 - 4.7.6 LR语法分析表的压缩
 - 4.7.7 4.7节的练习
- 4.8 使用二义性文法
 - 4.8.1 用优先级和结合性解决冲突
 - 4.8.2 “悬空-else”二义性
 - 4.8.3 LR语法分析中的错误恢复
 - 4.8.4 4.8节的练习
- 4.9 语法分析器的生成工具
 - 4.9.1 语法分析器的生成工具Yacc
 - 4.9.2 使用Yacc处理二义性文法
 - 4.9.3 用Lex创建Yacc的词法分析器
 - 4.9.4 Yacc中的错误恢复
 - 4.9.5 4.9节的练习
- 4.10 : 第4章的小结
- 4.11 第4章的参考文献
- 第5章 语法制导的翻译
 - 5.1 语法制导定义
 - 5.1.1 继承属性和综合属性
 - 5.1.2 在一棵语法分析树的结点上对一个SDD求值
 - 5.1.3 5.1节的练习
 - 5.2 SDD的求值顺序
 - 5.2.1 依赖图
 - 5.2.2 属性求值的顺序
 - 5.2.3 S-属性定义
 - 5.2.4 L-属性定义
 - 5.2.5 具有受控副作用的语义规则
 - 5.2.6 5.2节的练习
 - 5.3 语法制导翻译的应用
 - 5.3.1 抽象语法树的构造
 - 5.3.2 类型的结构
 - 5.3.3 5.3节的练习
 - 5.4 语法制导的翻译方案
 - 5.4.1 后缀翻译方案
 - 5.4.2 后缀SDT的语法分析栈实现
 - 5.4.3 产生式内部带有语义动作的SDT
 - 5.4.4 从SDT中消除左递归
 - 5.4.5 L-属性定义的SDT
 - 5.4.6 5.4节的练习
 - 5.5 实现L-属性的SDD

<<编译原理>>

5.5.1 在递归下降语法分析过程中进行翻译

5.5.2 边扫描边生成代码

5.5.3 L-属性的SDD和LL语法分析

5.5.4 L-属性的SDD的自底向上语法分析

5.5.5 5.5节的练习

5.6 第5章的总结

5.7 第5章的参考文献

第6章 中间代码生成

第7章 运行时刻环境

第7章 总结

第8章 代码生成

第9章 机器无关优化

第10章 指令级并行

第11章 并行性和局部性的优化

第12章 过程间分析

<<编译原理>>

章节摘录

插图：第二个目标是编译器应该有效提高很多输入程序的性能。

性能通常意味着程序执行的速度。

我们也希望能够尽可能降低生成代码的大小，在嵌入式系统中更是如此。

而对于移动设备的情况，尽量降低代码的能耗也是我们期待的。

在通常情况下，提高执行效率的优化也能够节约能耗。

除了性能，错误报告和调试等的可用性方面也是很重要的。

第三，我们需要使编译时间保持在较短的范围内，以支持快速的开发和调试周期。

当机器变得越来越快，这个要求会越来越容易达到。

开始时，一个程序经常在没有进行优化的情况下开发和调试。

这么做不仅可以降低编译时间，更重要的是未经优化的程序比较容易调试。

这是因为编译器引入的优化经常使得源代码和目标代码之间的关系变得模糊。

在编译器中开启优化有时会暴露出源程序中的新问题，因此需要对经过优化的代码再次进行测试。

因为可能需要额外的测试工作，有时会阻止人们在应用中使用优化技术，当应用的性能不很重要的时候更是如此。

最后，编译器是一个复杂的系统，我们必须使系统保持简单以保证编译器的设计和维护费用是可管理的。

我们可以实现的优化技术有无穷多种，而创建一个正确有效的优化过程需要相当大的工作量。

我们必须划分不同优化技术的优先级别，只实现那些可以对实践中遇到的源程序带来最大好处的技术。

因此，我们在研究编译器时不仅要学习如何构造一个编译器，还要学习解决复杂和开放性问题的的一般方法学。

在编译器开发中用到的方法涉及理论和实验。

在开始的时候，我们通常根据直觉确定有哪些重要的问题并把它们明确描述出来。

<<编译原理>>

编辑推荐

《编译原理(第2版)》是编译领域无可替代的经典著作，被广大计算机专业人士誉为"龙书"。

《编译原理(第2版)》上一版自1986年出版以来，被世界各地的著名高等院校和研究机构（包括美国哥伦比亚大学、斯坦福大学、哈佛大学、普林斯顿大学、贝尔实验室）作为本科生和研究生的编译原理课程的教材。

该书对我国高等计算机教育领域也产生了重大影响。

编译领域里程碑式的经典著作——龙书，20年后终于出版新版！

这是一个延绵30年的故事，这是一部关于龙书的传奇！

最新版本，增添两章节内容，使龙书地位更权威！

第2版对每一章都进行了全面的修订，以反映自上一版出版20多年来软件工程。

程序设计语言和计算机体系结构方面的发展对编译技术的影响。

《编译原理(第2版)》全面介绍了编译器的设计，并强调编译技术在软件设计和开发中的广泛应用。

每章中都包含大量的习题和丰富的参考文献。

1977年，Alfred V. Aho和Jeffrey D. Ullman合作出版了《Principles of Compiler Design》，封面是一位骑士和一只恐龙，那恐龙是绿色的，因此被称为龙书或绿龙书。

1986年，原来的两位作者加上Ravi Sethi，升级了前一《编译原理(第2版)》，书名改为《compilers: Principles, Techniques and Tools》，封面依然沿用骑士和恐龙，那恐龙是红色的，因此被称为龙书二或者红龙书。

又过了一个9年又一个9年，编译领域的巨无霸——龙书始终都没有升级。

终于在2006年底，龙书升级了。

作者又增加了Monica S. Lam，名字与龙书二相同，封面依然沿用恐龙和武士的设计，这次的龙是紫色的，因此被称为龙书三或者紫龙书。

版权说明

本站所提供下载的PDF图书仅提供预览和简介，请支持正版图书。

更多资源请访问:<http://www.tushu007.com>