

<<flex 与 bison>>

图书基本信息

书名：<<flex 与 bison>>

13位ISBN编号：9787564119324

10位ISBN编号：7564119322

出版时间：2010-1

出版时间：东南大学出版社

作者：John Levine

页数：271

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>

<<flex 与 bison>>

前言

Flex and bison are tools designed for writers of compilers and interpreters, although they are also useful for many applications that will interest noncompiler writers. Any application that looks for patterns in its input or has an input or command language is a good candidate for flex and bison. Furthermore, they allow for rapid application prototyping, easy modification, and simple maintenance of programs. To stimulate your imagination, here are a few things people have used flex and bison, or their predecessors lex and yacc, to develop: The desktop calculator `bc`; The tools `eqn` and `pic`, typesetting preprocessors for mathematical equations and complex pictures; Many other "domain-specific languages" targeted for a particular application; PCC, the Portable C Compiler used with many Unix systems; Flex itself; A SQL database language translator.

Scope of This Book

Chapter 1, *Introducing Flex and Bison*, gives an overview of how and why flex and bison are used to create compilers and interpreters and demonstrates some simple applications including a calculator built in flex and bison. It also introduces basic terms we use throughout the book.

Chapter 2, *Using Flex*, describes how to use flex. It develops flex applications that count words in files, handle multiple and nested input files, and compute statistics on C programs.

Chapter 3, *Using Bison*, gives a full example using flex and bison to develop a fully functional desktop calculator with variables, procedures, loops, and conditional expressions. It shows the use of abstract syntax trees (ASTs), powerful and easy-to-use data structures for representing parsed input.

Chapter 4, *Parsing SQL*, develops a parser for the MySQL dialect of the SQL relational database language.

<<flex 与 bison>>

内容概要

《flex 与 bison(影印版)》内容简介：如果你需要分析或处理Linux或Unix中的文本数据，这本有用的书籍就向你讲解了如何使用flex和bison迅速解决问题。

《flex与bison》被期待已久，是经典O'Reilly系列书籍《lex & yacc》的续篇。

在原书出版以来的近20年中，flex和bison已被证明比原来的Unix工具更可靠、更强大。

《flex与bison》一书涵盖了Linux和Unix程序开发中相同的重要核心功能，以及一些重要的新主题。你会找到适用于新手的修订教程和适用于高级用户的参考资料，以及对每个程序的基本用法的解释，并且运用它们创建简单、独立的应用程序。

有了《flex与bison》，你会发现这些灵活的工具提供的广泛用途。

包括的主题有：
· 正则表达式工具无法处理的地址语法挤压(address syntax crunching)
· 生成编译器和解释器，并运用大范围的文本处理功能
· 解释代码、配置文件或任何其他结构化的格式
· 学习关键编程技术，包括抽象语法树和符号表
· 用完整的示例代码实现一个完善的SQL语法
· 使用新的功能，如纯(可重入)词法分析器(lexer)和语法分析器(parser)、功能强大的JGLR分析器和C++的接口

<<flex 与 bison>>

作者简介

John Levine, Taughannock Networks的创始人，著有20余本技术书籍，其中包括《lex & yacc》和《qmail》，均为O'Reilly出版。

书籍目录

Preface 1. Introducing Flex and Bison Lexical Analysis and Parsing Regular Expressions and Scanning Our First Flex Program Programs in Plain Flex Putting Flex and Bison Together The Scanner as Coroutine Tokens and Values Grammars and Parsing BNF Grammars Bison 's Rule Input Language Compiling Flex and Bison Programs Together Ambiguous Grammars: Not Quite Adding a Few More Rules Flex and Bison vs. Handwritten Scanners and Parsers Exercises 2. Using Flex Regular Expressions Regular Expression Examples How Flex Handles Ambiguous Patterns Context-Dependent Tokens File I/O in Flex Scanners Reading Several Files The I/O Structure of a Flex Scanner Input to a Flex Scanner Flex Scanner Output Start States and Nested Input Files Symbol Tables and a Concordance Generator Managing Symbol Tables Using a Symbol Table C Language Cross-Reference Exercises 3. Using Bison How a Bison Parser Matches Its Input Shift/Reduce Parsing What Bison 's LALR(1) Parser Cannot Parse A Bison Parser Abstract Syntax Trees An Improved Calculator That Creates ASTs Literal Character Tokens Building the AST Calculator Shift/Reduce Conflicts and Operator Precedence When Not to Use Precedence Rules An Advanced Calculator Advanced Calculator Parser Calculator Statement Syntax Calculator Expression Syntax Top-Level Calculator Grammar Basic Parser Error Recovery The Advanced Calculator Lexer Reserved Words Building and Interpreting ASTs Evaluating Functions in the Calculator User-Defined Functions Using the Advanced Calculator Exercises 4. Parsing SQL A Quick Overview of SQL Relational Databases Manipulating Relations Three Ways to Use SQL SQL to RPN The Lexer Scanning SQL Keywords Scanning Numbers Scanning Operators and Punctuation Scanning Functions and Names Comments and Miscellany The Parser The Top-Level Parsing Rules SQL Expressions Select Statements Delete Statement Insert and Replace Statements Update Statement Create Database Create Table User Variables The Parser Routines The Makefile for the SQL Parser Exercises 5. A Reference for Flex Specifications Structure of a Flex Specification Definition Section Rules Section User Subroutines BEGIN C++ Scanners Context Sensitivity Left Context Right Context Definitions (Substitutions) ECHO Input Management Stdio File Chaining Input Buffers Input from Strings File Nesting input() YY_INPUT Flex Library Interactive and Batch Scanners Line Numbers and yylineno Literal Block Multiple Lexers in One Program Combined Lexers Multiple Lexers Options When Building a Scanner Portability of Flex Lexers Porting Generated C Lexers Reentrant Scanners Extra Data for Reentrant Scanners Access to Reentrant Scanner Data Reentrant Scanners, Nested Files, and Multiple Scanners Using Reentrant Scanners with Bison Regular Expression Syntax Metacharacters REJECT Returning Values from yylex() Start States unput() yyinput() yyunput() yyleng yyles() yylex() and YY_DECL yymore() yyrestart() yy_scan_string and yy_scan_buffer YY_USER_ACTION yywrap() 6. A Reference for Bison Specifications Structure of a Bison Grammar Symbols Definition Section Rules Section User Subroutines Section Actions Embedded Actions Symbol Types for Embedded Actions Ambiguity and Conflicts Types of Conflicts Shift/Reduce Conflicts Reduce/Reduce Conflicts %expect GLR Parsers Bugs in Bison Programs Infinite Recursion Interchanging Precedence Embedded Actions C++ Parsers %code Blocks End Marker Error Token and Error Recovery %destructor Inherited Attributes (\$) Symbol Types for Inherited Attributes %initial-action Lexical Feedback Literal Block Literal Tokens Locations %parse-param Portability of Bison Parsers Porting Bison Grammars Porting Generated C Parsers Libraries Character Codes Precedence and Associativity Declarations Precedence Associativity Precedence Declarations Using Precedence and Associativity to Resolve Conflicts Typical Uses of Precedence Recursive Rules Left and Right Recursion Rules Special Characters %start Declaration Symbol Values Declaring Symbol Types Explicit Symbol Types Tokens Token Numbers Token Values %type Declaration %union Declaration Variant and Multiple Grammars Combined Parsers Multiple Parsers Using %name-prefix or the -p Flag Lexers for Multiple Parsers Pure Parsers y.output Files Bison Library main() yyerror() YYABORT YYACCEPT YYBACKUP yyclearin yydebug and YYDEBUG YYDEBUG yydebug yyerrok YYERROR yyerror() yyparse() YYRECOVERING() 7. Ambiguities and Conflicts The Pointer Model and Conflicts Kinds of Conflicts Parser States Contents of name.output Reduce/Reduce Conflicts Shift/Reduce Conflicts Review of Conflicts in name.output Common Examples of Conflicts Expression Grammars IF/THEN/ELSE Nested List Grammar How Do You Fix the

<<flex 与 bison>>

Conflict? IF/THEN/ELSE (Shift/Reduce) Loop Within a Loop (Shift/Reduce) Expression Precedence (Shift/Reduce) Limited Lookahead (Shift/Reduce or Reduce/Reduce) Overlap of Alternatives (Reduce/Reduce) Summary Exercises 8. Error Reporting and Recovery Error Reporting Locations Adding Locations to the Parser Adding Locations to the Lexer More Sophisticated Locations with Filenames Error Recovery Bison Error Recovery Freeing Discarded Symbols Error Recovery in Interactive Parsers Where to Put Error Tokens Compiler Error Recovery Exercises 9. Advanced Flex and Bison Pure Scanners and Parsers Pure Scanners in Flex Pure Parsers in Bison Using Pure Scanners and Parsers Together A Reentrant Calculator GLR Parsing GLR Version of the SQL Parser C++ Parsers A C++ Calculator C++ Parser Naming A C++ Parser Interfacing a Scanner with a C++ Parser Should You Write Your Parser in C++ ? Exercises Appendix: SQL Parser Grammar and Cross-Reference Glossary Index

章节摘录

插图：A bison specification has the same three-part structure as a flex specification. (Flex copied its structure from the earlier lex, which copied its structure from yacc, the predecessor of bison.) The first section, the definition section, handles control information for the parser and generally sets up the execution environment in which the parser will operate. The second section contains the rules for the parser, and the third section is C code copied verbatim into the generated C program. Bison creates the C program by plugging pieces into a standard skeleton file. The rules are compiled into arrays that represent the state machine that matches the input tokens. The actions have the `SN` and `@N` values translated into C and then are put into a switch statement within `yyparse0` that runs the appropriate action each time there's a reduction. Some bits of the skeleton have multiple versions from which bison chooses depending on what options are in use; for example, if the parser uses the locations feature, it includes code to handle location data. In this chapter we take the simple calculator example from Chapter 1 and extend it significantly. First, we rewrite it to take advantage of some handy bison shortcuts and change it to produce a reusable data structure rather than computing the values on the fly. Later, we'll add more complex syntax for loops and functions and show how to implement them in a simple interpreter. One of the most powerful data structures used in compilers is an abstract syntax tree (AST). In Chapter 1 we saw a parse tree, a tree that has a node for every rule used to parse the input string. In most real grammars, there are rules that exist to manage grouping but that add no meaning to the program. In the calculator example, the rules `exp: term` and `term: factor` exist only to tell the parser the relative precedence of the operators. An AST is basically a parse tree that omits the nodes for the uninteresting rules.

<<flex 与 bison>>

媒体关注与评论

“我很高兴看到John彻底详尽地重写这本经典书。他更新的示例和说明，能够帮助老用户和新手摆脱模仿那些已经根深蒂固的旧lex和yacc。”
——Joel E. Denny bison维护人员

<<flex 与 bison>>

编辑推荐

《flex 与 bison(影印版)》是由东南大学出版社出版的。

<<flex 与 bison>>

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>