

自制编译器

自制编译器

How to Develop a Compiler

[日]青木峰郎 / 著 严圣迪 杨云 / 译

realme
真实世界
数字生活



中国工信出版集团

人民邮电出版社
POSTS & TELECOM PRESS

[自制编译器_下载链接1](#)

著者:[日] 青木峰郎

出版者:人民邮电出版社

出版时间:2016-6

装帧:平装

isbn:9787115422187

本书将带领读者从头开始制作一门语言的编译器。笔者特意为本书设计了Cb语言，Cb可以说是C语言的子集，实现了包括指针运算等在内的C语言的主要部分。本书所实现的编译器就是Cb语言的编译器，是实实在在的编译器，而非有诸多限制的玩具。另外，除编译器之外，本书对以编译器

为中心的编程语言的运行环境，即编译器、汇编器、链接器、硬件、运行时环境等都有所提及，介绍了程序运行的所有环节。

作者介绍:

作者简介:

青木峰郎

程序员，著有《Ruby程序设计268技（第2版）》《Ruby源代码完全解说》《Linux程序设计》等多部编程相关著作。并积极参与标准库维护、文档维护等各种各样的活动。

译者简介:

严圣逸

毕业于上海交通大学。8年软件开发经验，期间赴日本工作。现就职于想能信息科技（上海）有限公司，从事基于云平台的客户关系管理及各类营销自动化系统的开发工作。译有《高效团队开发：工具与方法》。

绝云

毕业于清华大学软件学院。曾在日本创意公司KAYAC从事即时通讯软件及社交游戏的开发工作，现任蚂蚁金服前端架构专家。译有《图解简单算法》等图书，曾参与《像外行一样思考，像专家一样实践（修订版）》的审校。

目录: 目录

第1章 开始制作编译器 1

1.1 本书的概要 2

本书的主题 2

本书制作的编译器 2

编译示例 2

可执行文件 3

编译 4

程序运行环境 6

1.2 编译过程 8

编译的4个阶段 8

语法分析 8

语义分析 9

生成中间代码 9

代码生成 10

优化 10

总结 10

1.3 使用Cb编译器进行编译 11

Cb编译器的必要环境 11

安装Cb编译器 11

Cb的Hello, World! 12

第2章 Cb和cbc 13

2.1 Cb语言的概要 14

Cb的Hello, World! 14

Cb中删减的功能 14

import 关键字 15

- 导入文件的规范 16
- 2.2 Cb编译器cbc 的构成 17
- cbc 的代码树 17
- cbc 的包 18
- compiler 包中的类群 18
- main 函数的实现 19
- commandMain 函数的实现 19
- Java5 泛型 20
- build 函数的实现 20
- Java 5 的foreach 语句 21
- compile 函数的实现 21
- 第1部分 代码分析
- 第3章 语法分析的概要 24
- 3.1 语法分析的方法 25
- 代码分析中的问题点 25
- 代码分析的一般规律 25
- 词法分析、语法分析、语义分析 25
- 扫描器的动作 26
- 单词的种类和语义值 27
- token 28
- 抽象语法树和节点 29
- 3.2 解析器生成器 30
- 什么是解析器生成器 30
- 解析器生成器的种类 30
- 解析器生成器的选择 31
- 3.3 JavaCC 的概要 33
- 什么是JavaCC 33
- 语法描述文件 33
- 语法描述文件的例子 34
- 运行JavaCC 35
- 启动JavaCC 所生成的解析器 36
- 中文的处理 37
- 第4章 词法分析 39
- 4.1 基于JavaCC 的扫描器的描述 40
- 本章的目的 40
- JavaCC 的正则表达式 40
- 固定字符串 41
- 连接 41
- 字符组 41
- 排除型字符组 41
- 重复1 次或多次 42
- 重复0 次或多次 42
- 重复n 次到m 次 42
- 正好重复n 次 43
- 可以省略 43
- 选择 43
- 4.2 扫描没有结构的单词 44
- TOKEN 命令 44
- 扫描标识符和保留字 44
- 选择匹配规则 45
- 扫描数值 46
- 4.3 扫描不生成token 的单词 48
- SKIP 命令和SPECIAL_TOKEN 命令 48
- 跳过空白符 48

- 跳过行注释 49
- 4.4 扫描具有结构的单词 50
- 最长匹配原则和它的问题 50
- 基于状态迁移的扫描 50
- MORE 命令 51
- 跳过块注释 52
- 扫描字符串字面量 53
- 扫描字符字面量 53
- 第5章 基于JavaCC 的解析器的描述 55
- 5.1 基于EBNF 语法的描述 56
- 本章的目的 56
- 基于JavaCC 的语法描述 56
- 终端符和非终端符 57
- JavaCC 的EBNF 表示法 58
- 连接 58
- 重复0 次或多次 59
- 重复1 次或多次 59
- 选择 60
- 可以省略 60
- 5.2 语法的二义性和token 的超前扫描 61
- 语法的二义性 61
- JavaCC 的局限性 62
- 提取左侧共通部分 63
- token 的超前扫描 63
- 可以省略的规则和冲突 64
- 重复和冲突 65
- 更灵活的超前扫描 66
- 超前扫描的相关注意事项 66
- 第6章 语法分析 68
- 6.1 定义的分析 69
- 表示程序整体的符号 69
- 语法的单位 69
- import 声明的语法 70
- 各类定义的语法 71
- 变量定义的语法 72
- 函数定义的语法 73
- 结构体定义和联合体定义的语法 74
- 结构体成员和联合体成员的语法 75
- typedef 语句的语法 76
- 类型的语法 76
- C 语言和C_b在变量定义上的区别 77
- 基本类型的语法 77
- 6.2 语句的分析 79
- 语句的语法 79
- if 语句的语法 80
- 省略if 语句和大括号 80
- while 语句的语法 81
- for 语句的语法 81
- 各类跳转语句的语法 82
- 6.3 表达式的分析 83
- 表达式的整体结构 83
- expr 的规则 83
- 条件表达式 84
- 二元运算符 85

- 6.4 项的分析 88
 - 项的规则 88
 - 前置运算符的规则 88
 - 后置运算符的规则 89
 - 字面量的规则 89
- 第2部分 抽象语法树和中间代码
- 第7章 JavaCC 的action 和抽象语法树 92
 - 7.1 JavaCC 的action 93
 - 本章的目的 93
 - 简单的action 93
 - 执行action 的时间点 93
 - 返回语义值的action 95
 - 获取终端符号的语义值 95
 - Token 类的属性 96
 - 获取非终端符号的语义值 98
 - 语法树的结构 99
 - 选择和action 99
 - 重复和action 100
 - 本节总结 102
 - 7.2 抽象语法树和节点 103
 - Node 类群 103
 - Node 类的定义 105
 - 抽象语法树的表示 105
 - 基于节点表示表达式的例子 107
- 第8章 抽象语法树的生成 110
 - 8.1 表达式的抽象语法树 111
 - 字面量的抽象语法树 111
 - 类型的表示 112
 - 为什么需要TypeRef 类 113
 - 一元运算的抽象语法树 114
 - 二元运算的抽象语法树 116
 - 条件表达式的抽象语法树 117
 - 赋值表达式的抽象语法树 118
 - 8.2 语句的抽象语法树 121
 - if 语句的抽象语法树 121
 - while 语句的抽象语法树 122
 - 程序块的抽象语法树 123
 - 8.3 声明的抽象语法树 125
 - 变量声明列表的抽象语法树 125
 - 函数定义的抽象语法树 126
 - 表示声明列表的抽象语法树 127
 - 表示程序整体的抽象语法树 128
 - 外部符号的import 128
 - 总结 129
 - 8.4 cbc 的解析器的启动 132
 - Parser 对象的生成 132
 - 文件的解析 133
 - 解析器的启动 134
- 第9章 语义分析 (1) 引用的消解 135
 - 9.1 语义分析的概要 136
 - 本章目的 136
 - 抽象语法树的遍历 137
 - 不使用Visitor 模式的抽象语法树的处理 137
 - 基于Visitor 模式的抽象语法树的处理 138

- Vistor 模式的一般化 140
- cbc 中Visitor 模式的实现 141
- 语义分析相关的cbc 的类 142
- 9.2 变量引用的消解 144
 - 问题概要 144
 - 实现的概要 144
 - Scope 树的结构 145
 - LocalResolver 类的属性 146
 - LocalResolver 类的启动 146
 - 变量定义的添加 147
 - 函数定义的处理 148
 - pushScope 方法 149
 - currentScope 方法 149
 - popScope 方法 150
 - 添加临时作用域 150
 - 建立VariableNode 和变量定义的关联 151
 - 从作用域树取得变量定义 151
- 9.3 类型名称的消解 153
 - 问题概要 153
 - 实现的概要 153
 - TypeResolver 类的属性 153
 - TypeResolver 类的启动 154
 - 类型的声明 154
 - 类型和抽象语法树的遍历 155
 - 变量定义的类型消解 156
 - 函数定义的类型消解 157
- 第10章 语义分析 (2) 静态类型检查 159
 - 10.1 类型定义的检查 160
 - 问题概要 160
 - 实现的概要 161
 - 检测有向图中的闭环的算法 162
 - 结构体、联合体的循环定义检查 163
 - 10.2 表达式的有效性检查 165
 - 问题概要 165
 - 实现的概要 165
 - DereferenceChecker 类的启动 166
 - SemanticError 异常的捕获 167
 - 非指针类型取值操作的检查 167
 - 获取非左值表达式地址的检查 168
 - 隐式的指针生成 169
 - 10.3 静态类型检查 170
 - 问题概要 170
 - 实现的概要 170
 - Сb中操作数的类型 171
 - 隐式类型转换 172
 - TypeChecker 类的启动 173
 - 二元运算符的类型检查 174
 - 隐式类型转换的实现 175
- 第11章 中间代码的转换 178
 - 11.1 cbc 的中间代码 179
 - 组成中间代码的类 180
 - 中间代码节点类的属性 181
 - 中间代码的运算符和类型 182
 - 各类中间代码 183

- 中间代码的意义 184
- 11.2 IRGenerator 类的概要 185
- 抽象语法树的遍历和返回值 185
- IRGenerator 类的启动 185
- 函数本体的转换 186
- 作为语句的表达式的判别 187
- 11.3 流程控制语句的转换 189
- if 语句的转换 (1) 概要 189
- if 语句的转换 (2) 没有else 部分的情况 190
- if 语句的转换 (3) 存在else 部分的情况 191
- while 语句的转换 191
- break 语句的转换 (1) 问题的定义 192
- break 语句的转换 (2) 实现的方针 193
- break 语句的转换 (3) 实现 194
- 11.4 没有副作用的表达式的转换 196
- UnaryOpNode 对象的转换 196
- BinaryOpNode 对象的转换 197
- 指针加减运算的转换 198
- 11.5 左值的转换 200
- 左边和右边 200
- 左值和右值 200
- cbc 中左值的表现 201
- 结构体成员的偏移 202
- 成员引用 (expr.memb) 的转换 203
- 左值转换的例外: 数组和函数 204
- 成员引用的表达式 (ptr->memb) 的转换 205
- 11.6 存在副作用的表达式的转换 206
- 表达式的副作用 206
- 有副作用的表达式的转换方针 206
- 简单赋值表达式的转换 (1) 语句 207
- 临时变量的引入 208
- 简单赋值表达式的转换 (2) 表达式 209
- 后置自增的转换 210
- 第3部分 汇编代码
- 第12章 x86 架构的概要 214
- 12.1 计算机的系统结构 215
- CPU 和存储器 215
- 寄存器 215
- 地址 216
- 物理地址和虚拟地址 216
- 各类设备 217
- 缓存 218
- 12.2 x86 系列CPU 的历史 220
- x86 系列CPU 220
- 32 位CPU 220
- 指令集 221
- IA-32 的变迁 222
- IA-32 的64 位扩展——AMD64 222
- 12.3 IA-32 的概要 224
- IA-32 的寄存器 224
- 通用寄存器 225
- 机器栈 226
- 机器栈的操作 227
- 机器栈的用途 227

- 栈帧 228
- 指令指针 229
- 标志寄存器 229
- 12.4 数据的表现形式和格式 231
 - 无符号整数的表现形式 231
 - 有符号整数的表现形式 231
 - 负整数的表现形式和二进制补码 232
- 字节序 233
- 对齐 233
- 结构体的表现形式 234
- 第13章 x86 汇编器编程 236
 - 13.1 基于GNU 汇编器的编程 237
 - GNU 汇编器 237
 - 汇编语言的Hello, World! 237
 - 基于GNU 汇编器的汇编代码 238
 - 13.2 GNU 汇编器的语法 240
 - 汇编版的Hello, World! 240
 - 指令 241
 - 汇编伪操作 241
 - 标签 241
 - 注释 242
 - 助记符后缀 242
 - 各种各样的操作数 243
 - 间接内存引用 244
 - x86 指令集的概要 245
 - 13.3 传输指令 246
 - mov 指令 246
 - push 指令和pop 指令 247
 - lea 指令 248
 - movsx 指令和movzx 指令 249
 - 符号扩展和零扩展 250
 - 13.4 算术运算指令 251
 - add 指令 251
 - 进位标志 252
 - sub 指令 252
 - imul 指令 252
 - idiv 指令和div 指令 253
 - inc 指令 254
 - dec 指令 255
 - neg 指令 255
 - 13.5 位运算指令 256
 - and 指令 256
 - or 指令 257
 - xor 指令 257
 - not 指令 257
 - sal 指令 258
 - sar 指令 258
 - shr 指令 259
 - 13.6 流程的控制 260
 - jmp 指令 260
 - 条件跳转指令 (jz、jnz、je、jne、……) 261
 - cmp 指令 262
 - test 指令 263
 - 标志位获取指令 (SETcc) 263

- call 指令 264
- ret 指令 265
- 第14章 函数和变量 266
 - 14.1 程序调用约定 267
 - 什么是程序调用约定 267
 - Linux/x86 下的程序调用约定 267
 - 14.2 Linux/x86 下的函数调用 269
 - 到函数调用完成为止 269
 - 到函数开始执行为止 270
 - 到返回原处理流程为止 271
 - 到清理操作完成为止 271
 - 函数调用总结 272
 - 14.3 Linux/x86 下函数调用的细节 274
 - 寄存器的保存和复原 274
 - caller-save 寄存器和callee-save 寄存器 274
 - caller-save 寄存器和callee-save 寄存器的灵活应用 275
 - 大数值和浮点数的返回方法 276
 - 其他平台的程序调用约定 277
- 第15章 编译表达式和语句 278
 - 15.1 确认编译结果 279
 - 利用cbc 进行确认的方法 279
 - 利用gcc 进行确认的方法 280
 - 15.2 x86 汇编的对象与DSL 282
 - 表示汇编的类 282
 - 表示汇编对象 283
 - 15.3 cbc 的x86 汇编DSL 285
 - 利用DSL 生成汇编对象 285
 - 表示寄存器 286
 - 表示立即数和内存引用 287
 - 表示指令 287
 - 表示汇编伪操作、标签和注释 288
 - 15.4 CodeGenerator 类的概要 290
 - CodeGenerator 类的字段 290
 - CodeGenerator 类的处理概述 290
 - 实现compileStmts 方法 291
 - cbc 的编译策略 292
 - 15.5 编译单纯的表达式 294
 - 编译Int 节点 294
 - 编译Str 节点 294
 - 编译Uni 节点(1) 按位取反 295
 - 编译Uni 节点(2) 逻辑非 297
 - 15.6 编译二元运算 298
 - 编译Bin 节点 298
 - 实现compileBinaryOp 方法 299
 - 实现除法和余数 300
 - 实现比较运算 300
 - 15.7 引用变量和赋值 301
 - 编译Var 节点 301
 - 编译Addr 节点 302
 - 编译Mem 节点 303
 - 编译Assign 节点 303
 - 15.8 编译jump 语句 305
 - 编译LabelStmt 节点 305
 - 编译Jump 节点 305

- 编译CJump 节点 305
- 编译Call 节点 306
- 编译Return 节点 307
- 第16章 分配栈帧 308
 - 16.1 操作栈 309
 - cbc 中的栈帧 309
 - 栈指针操作原则 310
 - 函数体编译顺序 310
 - 16.2 参数和局部变量的内存分配 312
 - 本节概述 312
 - 参数的内存分配 312
 - 局部变量的内存分配：原则 313
 - 局部变量的内存分配 314
 - 处理作用域内的局部变量 315
 - 对齐的计算 316
 - 子作用域变量的内存分配 316
 - 16.3 利用虚拟栈分配临时变量 318
 - 虚拟栈的作用 318
 - 虚拟栈的接口 319
 - 虚拟栈的结构 319
 - virtualPush 方法的实现 320
 - VirtualStack#extend 方法的实现 320
 - VirtualStack#top 方法的实现 321
 - virtualPop 方法的实现 321
 - VirtualStack#rewind 方法的实现 321
 - 虚拟栈的运作 322
 - 16.4 调整栈访问的偏移量 323
 - 本节概要 323
 - StackFrameInfo 类 323
 - 计算正在使用的callee-save 寄存器 324
 - 计算临时变量区域的大小 325
 - 调整局部变量的偏移量 325
 - 调整临时变量的偏移量 326
 - 16.5 生成函数序言和尾声 327
 - 本节概要 327
 - 生成函数序言 327
 - 生成函数尾声 328
 - 16.6 alloca 函数的实现 330
 - 什么是alloca 函数 330
 - 实现原则 330
 - alloca 函数的影响 331
 - alloca 函数的实现 331
- 第17章 优化的方法 333
 - 17.1 什么是优化 334
 - 各种各样的优化 334
 - 优化的案例 334
 - 常量折叠 334
 - 代数简化 335
 - 降低运算强度 335
 - 削除共同子表达式 335
 - 消除无效语句 336
 - 函数内联 336
 - 17.2 优化的分类 337
 - 基于方法的优化分类 337

- 基于作用范围的优化分类 337
- 基于作用阶段的优化分类 338
- 17.3 cbc 中的优化 339
- cbc 中的优化原则 339
- cbc 中实现的优化 339
- cbc 中优化的实现 339
- 17.4 更深层的优化 341
- 基于模式匹配选择指令 341
- 分配寄存器 342
- 控制流分析 342
- 大规模的数据流分析和SSA形式 342
- 总结 343
- 第4部分 链接和加载
- 第18章 生成目标文件 346
- 18.1 ELF 文件的结构 347
- ELF 的目的 347
- ELF 的节和段 348
- 目标文件的主要ELF节 348
- 使用readelf 命令输出节头 349
- 使用readelf 命令输出程序头 350
- 使用readelf 命令输出符号表 351
- readelf 命令的选项 351
- 什么是DWARF 格式 352
- 18.2 全局变量及其在ELF 文件中的表示 354
- 分配给任意ELF节 354
- 分配给通用ELF节 354
- 分配.bss节 355
- 通用符号 355
- 记录全局变量对应的符号 357
- 记录符号的附加信息 357
- 记录通用符号的附加信息 358
- 总结 358
- 18.3 编译全局变量 360
- generate 方法的实现 360
- generateAssemblyCode 方法的实现 360
- 编译全局变量 361
- 编译立即数 362
- 编译通用符号 363
- 编译字符串字面量 364
- 生成函数头 365
- 计算函数的代码大小 366
- 总结 366
- 18.4 生成目标文件 367
- as 命令调用的概要 367
- 引用GNUAssembler类 367
- 调用as 命令 367
- 第19章 链接和库 369
- 19.1 链接的概要 370
- 链接的执行示例 370
- gcc 和GNU ld 371
- 链接器处理的文件 372
- 常用库 374
- 链接器的输入和输出 374
- 19.2 什么是链接 375

- 链接时进行的处理 375
- 合并节 375
- 重定位 376
- 符号消解 377
- 19.3 动态链接和静态链接 379
- 两种链接方法 379
- 动态链接的优点 379
- 动态链接的缺点 380
- 动态链接示例 380
- 静态链接示例 381
- 库的检索规则 381
- 19.4 生成库 383
- 生成静态库 383
- Linux 中共享库的管理 383
- 生成共享库 384
- 链接生成的共享库 385
- 第20章 加载程序 387
- 20.1 加载ELF 段 388
- 利用mmap 系统调用进行文件映射 388
- 进程的内存镜像 389
- 内存空间的属性 390
- ELF 段对应的内存空间 390
- 和ELF 文件不对应的内存空间 392
- ELF 文件加载的实现 393
- 20.2 动态链接过程 395
- 动态链接加载器 395
- 程序从启动到终止的过程 395
- 启动ld.so 396
- 系统内核传递的信息 397
- AUX 矢量 397
- 读入共享库 398
- 符号消解和重定位 399
- 运行初始化代码 400
- 执行主程序 401
- 执行终止处理 402
- ld.so 解析的环境变量 402
- 20.3 动态加载 404
- 所谓动态加载 404
- Linux 下的动态加载 404
- 动态加载的架构 405
- 20.4 GNU ld 的链接 406
- 用于cbc 的ld 选项的结构 406
- C 运行时 407
- 生成可执行文件 408
- 生成共享库 408
- 第21章 生成地址无关代码 410
- 21.1 地址无关代码 411
- 什么是地址无关代码 411
- 全局偏移表 (GOT) 412
- 获取GOT 地址 412
- 使用GOT 地址访问全局变量 413
- 访问使用GOT 地址的文件内部的全局变量 414
- 过程链接表 (PLT) 414
- 调用PLT 入口 416

- 地址无关的可执行文件：PIE 416
- 21.2 全局变量引用的实现 418
- 获取GOT 地址 418
- PICThunk 函数的实现 418
- 删除重复函数并设置不可见属性 419
- 加载GOT 地址 420
- locateSymbols 函数的实现 421
- 全局变量的引用 421
- 访问全局变量：地址无关代码的情况下 422
- 函数的符号 423
- 字符串常量的引用 424
- 21.3 链接器调用的实现 425
- 生成可执行文件 425
- generateSharedLibrary 方法 426
- 21.4 从程序解析到执行 428
- build 和加载的过程 428
- 词法分析 429
- 语法分析 429
- 生成中间代码 430
- 生成代码 431
- 汇编 432
- 生成共享库 432
- 生成可执行文件 433
- 加载 433
- 第22章 扩展阅读 434
- 22.1 参考书推荐 435
- 编译器相关 435
- 语法分析相关 435
- 汇编语言相关 436
- 22.2 链接、加载相关 437
- 22.3 各种编程语言的功能 438
- 异常封装相关的图书 438
- 垃圾回收 438
- 垃圾回收相关的图书 439
- 面向对象编程语言的实现 439
- 函数式语言 440
- 附录 441
- A.1 参考文献 442
- A.2 在线资料 444
- A.3 源代码 445
- • • • • ([收起](#))

[自制编译器 下载链接1](#)

标签

编译原理

编译器

计算机

编译

编程

自制系列

计算机科学

compiler

评论

作者讲述的思路很好，编译器的基本实现流程都说到了。但可惜的Cb是用java实现的，大量的类和继承让代码显得臃肿拖沓，而即使采用了这么多类，这么大的代码量却仍然还有C的大量特性没有实现。对比之前看过的另外一个实现C11标准的编译器8cc，采用C11实现，完成基本C11特性的基础上却只有不到6000行代码量，并且代码简洁易懂，可读性和参考价值颇高。综上，这本书只能用来了解编译器的基本实现流程，代码我觉得就不用过于深究，可读性和研究的价值不大。个人见解，请勿拍砖。

我明白了！明白了！明白了！

加载和链接。Linux系统下通过mmap系统调用把程序加载到内存中。mmap是把文件内容映射到内存空间中的系统调用。所谓“映射”，意思是可以通过读取内存直接获得文件内容，也可以通过写内存对文件内容进行变更。

好看的飞起~~~，最近正在搞JavaCC 读到 IR 就不读了 书确实不错

整本书对 javaCC 的依赖很强，不是很推荐

词法语法分析 生成中间代码 汇编 链接 生成可执行文件 --- 也了解jvm的原理了

这也能叫做书?

后端比前端复杂多了 (得补补汇编了

毕竟也做过编译器，所以买了一本看架构，没有读完。

用来大概了解编译流程挺合适，代码太多，不实际看源码一会就晕了。汇编部分没基础，跳过了

缺少对于自制编译器技能树的描述，或者说开头没有总览来介绍书籍各部分的功能及作用，前后章节的关联性也很差。因此对新手来说，刚开始读会变的很迷茫，无法将各部分内容与自制编译器整个过程联系起来。

写的什么东西

java写的，不适合我

自制系列

理论居多，基本上在书中找不到可运行的代码，导致学习起来有点迷茫，好像看了半天却啥都做不出来。相比之下《两周自制脚本语言》写得注重实践部分，并且把难点放在附录中做解释。

[自制编译器_下载链接1](#)

书评

[自制编译器_下载链接1](#)