华章科技

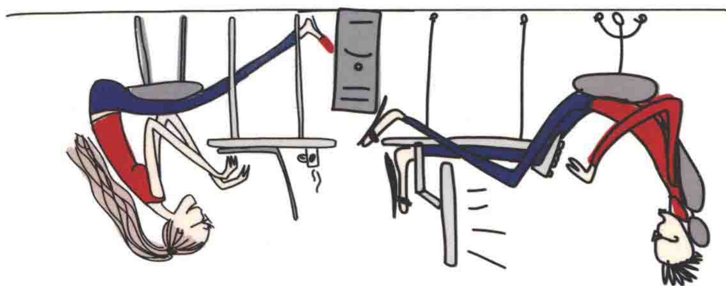
HZ BOOKS

自己动手系列

自己动手构造编译系统

编译、汇编与链接

范志东 张琼声 著



机械工业出版社

China Machine Press

自己动手构造编译系统

编译、汇编与链接

范志东 张琼声 著

图书在版编目（CIP）数据

自己动手构造编译系统：编译、汇编与链接 / 范志东，张琼声著．—北京：机械工业出版社，2016.7
（自己动手系列）

ISBN 978-7-111-54355-8

I. 自… II. ①范… ②张… III. 编译器 IV. TP314

中国版本图书馆 CIP 数据核字（2016）第 163077 号

自己动手构造编译系统：编译、汇编与链接

出版发行：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：佘 洁	责任校对：董纪丽
印 刷：中国电影出版社印刷厂	版 次：2016 年 8 月第 1 版第 1 次印刷
开 本：186mm×240mm 1/16	印 张：19
书 号：ISBN 978-7-111-54355-8	定 价：69.00 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）88379426 88361066　　投稿热线：（010）88379604

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

版权所有•侵权必究

封底无防伪标均为盗版

本书法律顾问：北京大成律师事务所 韩光 / 邹晓东

Foreword 序

小范从本科毕业设计开始写编译器的实现代码，为他选择这个题目的初衷是希望把编译系统与操作系统、计算机体系结构相关的结合点找出来、弄清楚，为教学提供可用的实例。本科毕业设计结束时小范完成了一个最简单的C语言子集的编译器，生成的汇编程序经过汇编和链接后可以正确执行。研究生期间我们决定继续编译系统实现技术方向的研究工作，主要完成汇编器和链接器这两大模块。小范用一颗好奇、求知的心指引自己，利用一切可以搜集到的资料，用“日拱一卒”的劲头一步一步接近目标。每天的日子都可能有不同的“干扰”——名企的实习、发论文、做项目、参加竞赛、考认证，身边的同学在快速积攒各种经历和成果的时候，小范要保持内心的平静，专注于工作量巨大而是否有回报还未曾可知的事情。三年的时间里，没有奖学金，没有项目经费，有的是没完没了的各种问题，各种要看的书、资料和要完成的代码，同时还要关注大数据平台、编程语言等新技术的发展。

“汇编器完成了”“链接器完成了”，好消息接踵而至。小范说，“把编译器的代码重写一下，加上代码优化吧？”我说“好”，其实，这个“好”说起来容易，而小范那里增加的工作量可想而知，这绝不是那么轻松的事情。优化的基本原理有了，怎么设计算法来实现呢？整个编译器的文法比本科毕业设计时扩充了很多。编译器重写、增加代码优化模块、完成汇编器和链接器，难度和工作量可想而知。每当小范解决一个问题，完成一个功能，就会非常开心地与我分享。看小范完成的一行行规范、漂亮的代码，听他兴奋地讲解，很难说与听郎朗的钢琴协奏曲《黄河之子》、德沃夏克的《自新大陆》比哪一个更令人陶醉，与听交响曲《嘎达梅林》比哪一个更令人震撼。当小范完成链接器后，我说：“小范，写书吧，不写下来太可惜了。”就这样，小范再次如一辆崭新的装甲车，轰隆前行，踏上了笔耕不辍的征程。2015年暑假，细读和修改这部30多万字的书稿，感慨万千，完成编译系统的工作量、四年的甘苦与共、超然物外的孤独都在这字里行间跳跃。写完这部原创书对一个年轻学生来说是极富挑战

的，但是他完成了，而且完成得如此精致、用心。

小范来自安徽的农村，面对生活中的各种困惑、困难，他很少有沮丧、悲观的情绪，永远有天然的好奇心，保留着顽童的天真、快乐与坦率。他开始写本书时23岁，完成全书的初稿时25岁。写编译系统和操作系统内核并非难以企及，只是需要一份淡然、专注和坚持。

如果你想了解计算机是如何工作的，为什么程序会出现不可思议的错误？高级语言程序是如何被翻译成机器语言代码的？编译器在程序的优化方面能做哪些工作？软件和硬件是怎么结合工作的？各种复杂的数据结构和算法，包括图论在实现编译系统时如何应用？有限自动机在词法分析中的作用是什么？其程序又如何实现？那么本书可以满足你的好奇心和求知欲。如何实现编译系统？如何实现编译器？如何实现汇编器？如何使用符号表？如何结合操作系统加载器的需要实现链接器？ Intel 的指令是如何构成的？如何实现不同的编译优化算法？对这些问题，本书结合作者实现的代码实例进行了详尽的阐述，对提高程序员的专业素质有实际的助益，同时本书也可以作为计算机科学相关专业教师的参考书和编译原理实习类课程的教材。

2013年在新疆参加全国操作系统和组成原理教学研讨会时，我带着打印出来的两章书稿给了机械工业出版社的温莉芳老师，与她探讨这本书出版的意义和可行性，她给了我们很大的鼓励和支持，促成了本书的完成。在此，特别感谢温莉芳老师。

本书的责任编辑佘洁老师与作者反复沟通，对本书进行了认真、耐心的编辑，感谢她的辛勤付出。

中国石油大学（华东）的李村合老师在编译器设计的初期给予了我们指导和建议。马力老师在繁忙的工作之余，认真审阅书稿，给出了详细的修改意见。王小云、程坚、梁红卫、葛永文老师对本书提出了他们的意见，并给出了认真的评价。赵国梁同学对书中的代码和文字做了细心的校对。在此，对他们表示衷心的感谢。最后要感谢小范勤劳、坚韧的爸爸妈妈，是他们一直给予他无私的支持和持续的鼓励。

感恩所有给予我们帮助和鼓励的老师、同学和朋友！

张琼声

2016年春于北京

试读结束：需要全本请在线购买： www.ertongbook.com

Preface 前　言

本书适合谁读

本书是一本描述编译系统实现的书籍。这里使用“编译系统”一词，主要是为了与市面上描述编译器实现的书籍进行区分。本书描述的编译系统不仅包含编译器的实现，还包括汇编器、链接器的实现，以及机器指令与可执行文件格式的知识。因此，本书使用“编译系统”一词作为编译器、汇编器和链接器的统称。

本书的目的是希望读者能通过阅读本书清晰地认识编译系统的工作流程，并能自己尝试构造一个完整的编译系统。为了使读者更容易理解和学习编译系统的构造方法，本书将描述的重点放在编译系统的关键流程上，并对工业化编译系统的实现做了适当的简化。如果读者对编译系统实现的内幕感兴趣，或者想自己动手实现一个编译系统的话，本书将非常适合你阅读。

阅读本书，你会发现书中的内容与传统的编译原理教材以及描述编译器实现的书籍有所不同。本书除了描述一个编译器的具体实现外，还描述了一般书籍较少涉及的汇编器和链接器的具体实现。而且本书并非“纸上谈兵”，在讲述每个功能模块时，书中都会结合具体实现代码来阐述模块功能的实现。通过本书读者将会学习如何使用有限自动机构造词法分析器，如何将文法分析算法应用到语法分析过程，如何使用数据流分析进行中间代码的优化，如何生成合法的汇编代码，如何产生二进制指令信息，如何在链接器内进行符号解析和重定位，如何生成目标文件和可执行文件等。

本书的宗旨是为意欲了解或亲自实现编译系统的读者提供指导和帮助。尤其是计算机专业的读者，通过自己动手写出一个编译系统，能加强读者对计算机系统从软件层次到硬件层次的理解。同时，深入挖掘技术幕后的秘密也是对专业兴趣的一种良好培养。GCC 本身是一套非常完善的工业化编译系统（虽然我们习惯上称它为编译器），然而单凭个人之力无法做到

像 GCC 这样完善，而且很多时候是没有必要做出一个工程化的编译器的。本书试图帮助读者深入理解编译的过程，并能按照书中的指导实现一个能正常工作的编译器。在自己亲自动手实现一个编译系统的过程中，读者获得的不仅仅是软件开发的经历。在开发编译系统的过程中，读者还会学习很多与底层相关的知识，而这些知识在一般的专业教材中很少涉及。

如果读者想了解计算机程序底层工作的奥秘，本书能够解答你内心的疑惑。如果读者想自定义一种高级语言，并希望使该语言的程序在计算机上正常运行，本书能帮助你较快地达到目的。如果读者想从实现一个编译器的过程中，加强对编译系统工作流程的理解，并尝试深入研究 GCC 源码，本书也能为你提供很多有价值的参考。

基础知识储备

本书尽可能地不要求读者有太多的基础知识准备，但是编译理论属于计算机学科比较深层次的知识领域，难免对读者的知识储备有所要求。本书的编译系统是基于 Linux x86 平台实现的，因此要求读者对 Linux 环境的 C/C++ 编程有所了解。另外，理解汇编器的实现内容需要读者对 x86 的汇编指令编程比较熟悉。本书不会描述过多编译原理教材中涉及的内容，所以要求读者具备编译原理的基础知识。不过读者不必过于担心，本书会按照循序渐进的方式描述编译系统的实现，在具体的章节中会将编译系统实现的每个细节以及所需的知识阐述清楚。

本书内容组织

本书共 7 章，各章的主要内容分别如下。

第 1 章　代码背后

从程序设计开始，追溯代码背后的细节，引出编译系统的概念。

第 2 章　编译系统设计

按照编译系统的工作流程，介绍本书编译系统的设计结构。

第 3 章　编译器构造

描述如何使用有限自动机识别自定义高级语言的词法记号，如何使用文法分析算法识别程序的语法模块，如何对高级语言上下文相关信息进行语义合法性检查，如何使用语法制导翻译进行代码生成，以及编译器工作时符号信息的管理等。

第 4 章　编译优化

介绍中间代码的设计和生成，如何利用数据流分析实现中间代码优化，如何对变量进行寄存器分配，目标代码生成阶段如何使用窥孔优化器对目标代码进行优化。

第 5 章　二进制表示

描述 Intel x86 指令的基本格式，并将 AT&T 汇编与 Intel 汇编进行对比。描述 ELF 文件的基本格式，介绍 ELF 文件的组织和操作方法。

第 6 章　汇编器构造

描述汇编器词法分析和语法分析的实现，介绍汇编器如何提取目标文件的主要表信息，并描述 x86 二进制指令的输出方法。

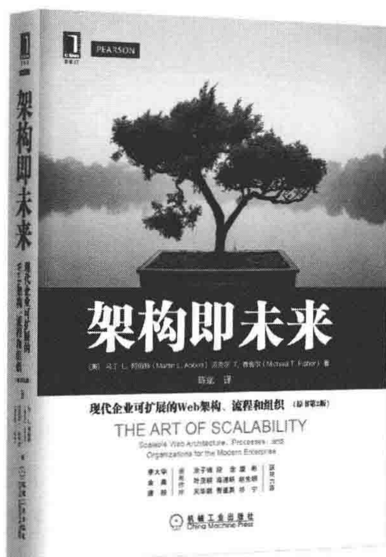
第 7 章　链接器构造

介绍如何为可重定位目标文件的段进行地址空间分配，描述链接器符号解析的流程，以及符号地址的计算方法，并介绍重定位在链接器中的实现。

随书源码

本书实现的编译系统代码已经托管到 github，源码可以使用 GCC 5.2.0 编译通过。代码的 github 地址是 https://github.com/fanzhidongyzby/cit。代码分支 x86 实现了基于 Intel x86 体系结构的编译器、汇编器和链接器，编译系统生成的目标文件和可执行文件都是 Linux 下标准的 ELF 文件格式。代码分支 arm 实现了基于 ARM 体系结构的编译器，目前支持生成 ARM 7 的汇编代码。

推荐阅读



架构即未来：现代企业可扩展的Web架构、流程和组织（原书第2版）

作者：马丁 L. 阿伯特 等 ISBN：978-7-111-53264-4 定价：99.00元

互联网技术管理与架构设计的"孙子兵法"

跨越横亘在当代商业增长和企业IT系统架构之间的鸿沟

有胆识的商业高层人士必读经典

李大学、余晨、唐毅 亲笔作序 涂子沛、段念、唐彬等 联合力荐

任何一个持续成长的公司最终都需要解决系统、组织和流程的扩展性问题。本书汇聚了作者从eBay、VISA、Salesforce.com到Apple超过30年的丰富经验，全面阐释了经过验证的信息技术扩展方法，对所需要掌握的产品和服务的平滑扩展做了详尽的论述，并在第1版的基础上更新了扩展的策略、技术和案例。

针对技术和非技术的决策者，马丁·阿伯特和迈克尔·费舍尔详尽地介绍了影响扩展性的各个方面，包括架构、过程、组织和技术。通过阅读本书，你可以学习到以最大化敏捷性和扩展性来优化组织机构的新策略，以及对云计算（IaaS/PaaS）、NoSQL、DevOps和业务指标等的新见解。而且利用其中的工具和建议，你可以系统化地清除扩展性道路上的障碍，在技术和业务上取得前所未有的成功。

目　录 Contents

序

前言

第1章　代码背后 …… 1

1.1　从编程聊起 …… 1

1.2　历史渊源 …… 2

1.3　GCC的工作流程 …… 3

1.3.1　预编译 …… 4

1.3.2　编译 …… 5

1.3.3　汇编 …… 6

1.3.4　链接 …… 7

1.4　设计自己的编译系统 …… 8

1.5　本章小结 …… 9

第2章　编译系统设计 …… 11

2.1　编译程序的设计 …… 11

2.1.1　词法分析 …… 12

2.1.2　语法分析 …… 13

2.1.3　符号表管理 …… 14

2.1.4　语义分析 …… 15

2.1.5　代码生成 …… 16

2.1.6　编译优化 …… 16

2.2　x86指令格式 …… 18

2.3　ELF文件格式 …… 19

2.4　汇编程序的设计 …… 21

2.4.1　汇编词法、语法分析 …… 22

2.4.2　表信息生成 …… 23

2.4.3　指令生成 …… 24

2.5　链接程序的设计 …… 25

2.5.1　地址空间分配 …… 25

2.5.2　符号解析 …… 26

2.5.3　重定位 …… 27

2.6　本章小结 …… 27

第3章　编译器构造 …… 29

3.1　词法分析 …… 29

3.1.1　扫描器 …… 30

3.1.2　词法记号 …… 32

3.1.3　有限自动机 …… 36

3.1.4　解析器 …… 40

3.1.5　错误处理 …… 53

3.2 语法分析 ………………………… 55
3.2.1 文法定义 ………………………… 55
3.2.2 递归下降子程序 ………………… 65
3.2.3 错误处理 ………………………… 70
3.3 符号表管理 ……………………… 74
3.3.1 符号表数据结构 ………………… 75
3.3.2 作用域管理 ……………………… 78
3.3.3 变量管理 ………………………… 82
3.3.4 函数管理 ………………………… 88
3.4 语义分析 ………………………… 93
3.4.1 声明与定义语义检查 …………… 93
3.4.2 表达式语义检查 ………………… 95
3.4.3 语句语义检查 …………………… 97
3.4.4 错误处理 ………………………… 98
3.5 代码生成 …………………………101
3.5.1 中间代码设计 ……………………102
3.5.2 程序运行时存储 …………………105
3.5.3 函数定义与 return 语句翻译 …108
3.5.4 表达式翻译 ………………………110
3.5.5 复合语句与 break、continue 语句翻译 ………………………120
3.5.6 目标代码生成 ……………………132
3.5.7 数据段生成 ………………………141
3.6 本章小结 …………………………145

第 4 章 编译优化 ……………………147

4.1 数据流分析 ………………………149
4.1.1 流图 ………………………………149
4.1.2 数据流分析框架 …………………152
4.2 中间代码优化 ……………………155
4.2.1 常量传播 …………………………155
4.2.2 复写传播 …………………………167
4.2.3 死代码消除 ………………………172
4.3 寄存器分配 ………………………177
4.3.1 图着色算法 ………………………177
4.3.2 变量栈帧偏移计算 ………………182
4.4 窥孔优化 …………………………187
4.5 本章小结 …………………………190

第 5 章 二进制表示 ………………191

5.1 x86 指令 …………………………191
5.1.1 指令前缀 …………………………192
5.1.2 操作码 ……………………………194
5.1.3 ModR/M 字段 ……………………196
5.1.4 SIB 字段 …………………………198
5.1.5 偏移 ………………………………201
5.1.6 立即数 ……………………………201
5.1.7 AT&T 汇编格式 …………………202
5.2 ELF 文件 …………………………204
5.2.1 文件头 ……………………………205
5.2.2 段表 ………………………………207
5.2.3 程序头表 …………………………209
5.2.4 符号表 ……………………………213
5.2.5 重定位表 …………………………214
5.2.6 串表 ………………………………215
5.3 本章小结 …………………………217

第 6 章 汇编器构造 ………………219

6.1 词法分析 …………………………220
6.1.1 词法记号 …………………………220
6.1.2 有限自动机 ………………………222
6.2 语法分析 …………………………223

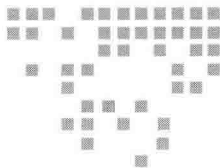
试读结束：需要全本请在线购买： www.ertongbook.com

6.2.1 汇编语言程序 ……223
6.2.2 数据定义 ……225
6.2.3 指令 ……226
6.3 符号表管理 ……227
6.3.1 数据结构 ……228
6.3.2 符号管理 ……230
6.4 表信息生成 ……234
6.4.1 段表信息 ……235
6.4.2 符号表信息 ……238
6.4.3 重定位表信息 ……239
6.5 指令生成 ……246
6.5.1 双操作数指令 ……247
6.5.2 单操作数指令 ……251
6.5.3 零操作数指令 ……254
6.6 目标文件生成 ……255
6.7 本章小结 ……261

第7章 链接器构造 ……263

7.1 信息收集 ……264
7.1.1 目标文件信息 ……264
7.1.2 段数据信息 ……266
7.1.3 符号引用信息 ……268
7.2 地址空间分配 ……269
7.3 符号解析 ……272
7.3.1 符号引用验证 ……274
7.3.2 符号地址解析 ……276
7.4 重定位 ……277
7.5 程序入口点与运行时库 ……281
7.6 可执行文件生成 ……283
7.7 本章小结 ……290

参考文献 ……291



第 1 章 Chapter 1

代码背后

知其然，并知其所以然。

——《朱子语类》

1.1 从编程聊起

说起编程，如果有人问我们敲进计算机的第一段代码是什么，相信很多人会说出同一个答案——“Hello World !”。编程语言的教材一般都会把这段代码作为书中的第一个例子呈现给读者。当我们按照课本或者老师的要求把它输入到开发环境，然后单击“编译”和“运行”按钮，映入眼帘的那行字符串定会令人欣喜不已！然而激动过后，一股强烈的好奇心可能会驱使我们去弄清一个新的概念——编译是什么？

遗憾的是，一般教授编程语言的老师不会介绍太多关于它的内容，最多会告诉我们：代码只有经过编译，才能在计算机中正确执行。随着知识和经验的不断积累，我们逐渐了解到当初单击“编译”按钮的时候，计算机在幕后做了一系列的工作。它先对源代码进行编译，生成二进制目标文件，然后对目标文件进行链接，最后生成一个可执行文件。即便如此，我们对编译的流程也只有一个模糊的认识。

直到学习了编译原理，才发现编译器原来就是语言翻译程序，它把高级语言程序翻译成低级汇编语言程序。而汇编语言程序是不能被计算机直接识别的，必须靠汇编器把它翻译为计算机硬件可识别的机器语言程序。而根据之前对目标文件和链接器的了解，我们可能猜测到机器语言应该是按照二进制的形式存储在目标文件内部的。可是目标文件到底包含什么，

链接后的可执行文件里又有什么？问题貌似越来越多。

图 1-1 展示了编译的大致工作流程，相信拥有一定编程经验的人，对该图所表达的含义并不陌生。为了让源代码能正常地运行在计算机上，计算机对代码进行了“繁复”的处理。可是，编译器既然是语言翻译程序，为什么不把源代码直接翻译成机器语言，却还要经过汇编和链接的过程呢?

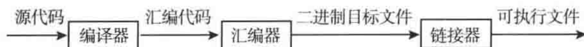


图 1-1 编译的流程

似乎我们解决了一些疑惑后，总是会有更多的疑惑接踵而来。但也正是这些层出不穷的疑惑，促使我们不断地探究简单问题背后的复杂机制。当挖掘出这些表象下覆盖的问题本质时，可能比首次敲出“Hello World!”程序时还要喜悦。在后面的章节中，将会逐步探讨编译背后的本质，将谜团一一揭开，最终读者自己可动手构造出本书所实现的编译系统——编译器、汇编器与链接器，真正做到“知其然，并知其所以然”。

1.2 历史渊源

历史上很多新鲜事物的出现都不是偶然的，计算机学科的技术和知识如此，编译系统也不例外，它的产生来源于编程工作的需求。编程本质上是人与计算机交流，人们使用计算机解决问题，必须把问题转化为计算机所能理解的方式。当问题规模逐渐增大时，编程的劳动量自然会变得繁重。编译系统的出现在一定程度上降低了编程的难度和复杂度。

在计算机刚刚诞生的年代，人们只能通过二进制机器指令指挥计算机工作，计算机程序是依靠人工拨动计算机控制面板上的开关被输入到计算机内部的。后来人们想到使用穿孔卡片来代替原始的开关输入，用卡片上穿孔的有无表示计算机世界的“0”和“1”，让计算机自动读取穿孔卡片实现程序的录入，这里录入的指令就是常说的二进制代码。然而这种编程工作在现在看起来简直就是一个“噩梦”，因为一旦穿孔卡片的制作出现错误，所有的工作都要重新来过。

人们很快就发现了使用二进制代码控制计算机的不足，因为人工输入二进制指令的错误率实在太高了。为了解决这个问题，人们用一系列简单明了的助记符代替计算机的二进制指令，即我们熟知的汇编语言。可是计算机只能识别二进制指令，因此需要一个已有的程序自动完成汇编语言到二进制指令的翻译工作，于是汇编器就产生了。程序员只需要写出汇编代码，然后交给汇编器进行翻译，生成二进制代码。因此，汇编器将程序员从烦琐的二进制代码中解脱出来。

使用汇编器提高了编程的效率，使得人们有能力处理更复杂的计算问题。随着计算问题复杂度的提高，编程中出现了大量的重复代码。人们不愿意进行重复的劳动，于是就想办法将公共的代码提取出来，汇编成独立的模块存储在目标文件中，甚至将同一类的目标文件打

包成库。由于原本写在同一个文件内的代码被分割到多个文件中，那么最终还需要将这些分离的文件拼装起来形成完整的可执行代码。但是事情并没有那么简单，由于文件的模块化分割，文件间的符号可能会相互引用。人们需要处理这些引用关系，重新计算符号的引用地址，这就是链接器的基本功能。链接器使得计算机能自动把不同的文件模块准确无误地拼接起来，使得代码的复用成为可能。

图 1-2 描述的链接方式称为静态链接，但这种方式也有不足之处。静态链接器把公用库内的目标文件合并到可执行文件内部，使得可执行文件的体积变得庞大。这样做会导致可执行文件版本难以更新，也导致了多个程序加载后相同的公用库代码占用了多份内存空间。为了解决上述的问题，现代编译系统都引入了动态链接方式（见图 1-3）。动态链接器不会把公用库内的目标文件合并到可执行文件内，而仅仅记录动态链接库的路径信息。它允许程序运行前才加载所需的动态链接库，如果该动态链接库已加载到内存，则不需要重复加载。另外，动态链接器也允许将动态链接库的加载延迟到程序执行库函数调用的那一刻。这样做，不仅节约了磁盘和内存空间，还方便了可执行文件版本的更新。如果应用程序模块设计合理的话，程序更新时只需要更新模块对应的动态链接库即可。当然，动态链接的方式也有缺点。运行时链接的方式会增加程序执行的时间开销。另外，动态链接库的版本错误可能会导致程序无法执行。由于静态链接和动态链接的基本原理类似，且动态链接器的实现相对复杂，因此本书编译系统所实现的链接器采用静态链接的方式。

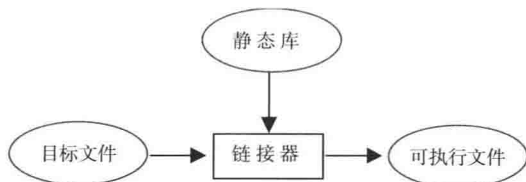


图 1-2　静态链接

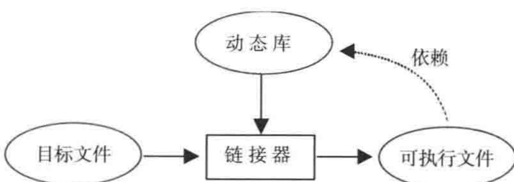


图 1-3　动态链接

汇编器和链接器的出现大大提高了编程效率，降低了编程和维护的难度。但是人们对汇编语言的能力并不满足，有人设想要是能像写数学公式那样对计算机编程就太方便了，于是就出现了如今形形色色的高级编程语言。这样就面临与当初汇编器产生时同样的问题——如何将高级语言翻译为汇编语言，这正是编译器所做的工作。编译器比汇编器复杂得多。汇编语言的语法比较单一，它与机器语言有基本的对应关系。而高级语言形式比较自由，计算机识别高级语言的含义比较困难，而且它的语句翻译为汇编语言序列时有多种选择，如何选择更好的序列作为翻译结果也是比较困难的，不过最终这些问题都得以解决。高级语言编译器的出现，实现了人们使用简洁易懂的编程语言与计算机交流的目的。

1.3　GCC 的工作流程

在着手构造编译系统之前，需要先介绍编译系统应该做的事情，而最具参考价值的资料

就是主流编译器的实现。GNU 的 GCC 编译器是工业化编译器的代表，因此我们先了解 GCC 都在做什么。

我们写一个最简单的“HelloWorld”程序，代码存储在源文件 hello.c 中，源文件内容如下：

```
#include<stdio.h>
int main()
{
    printf("Hello World!");
    return 0;
}
```

如果将 hello.c 编译并静态链接为可执行文件，使用如下 gcc 命令直接编译即可：

```
$gcc hello.c -o hello -static
```

hello 即编译后的可执行文件。

如果查看 GCC 背后的工作流程，可以使用--verbose 选项。

```
$gcc hello.c -o hello -static --verbose
```

输出的信息如下：

```
$cc1 -quiet hello.c -o hello.s
$as -o hello.o hello.s
$collect2 -static -o hello \
    crt1.o crti.o crtbeginT.o hello.o \
    --start-group libgcc.a libgcc_eh.a libc.a --end-group \
    crtend.o crtn.o
```

为了保持输出信息的简洁，这里对输出信息进行了整理。可以看出，GCC 编译背后使用了 cc1、as、collect2 三个命令。其中 cc1 是 GCC 的编译器，它将源文件 hello.c 编译为 hello.s。as 是汇编器命令，它将 hello.s 汇编为 hello.o 目标文件。collect2 是链接器命令，它是对命令 ld 的封装。静态链接时，GCC 将 C 语言运行时库（CRT）内的 5 个重要的目标文件 crt1.o、crti.o、crtbeginT.o、crtend.o、crtn.o 以及 3 个静态库 libgcc.a、libgcc_eh.a、libc.a 链接到可执行文件 hello。此外，cc1 在对源文件编译之前，还有预编译的过程。

因此，我们从预编译、编译、汇编和链接四个阶段查看 GCC 的工作细节。

1.3.1 预编译

GCC 对源文件的第一阶段的处理是预编译，主要是处理宏定义和文件包含等信息。命令格式如下：

```
$gcc -E hello.c -o hello.i
```

预编译器将 hello.c 处理后输出到文件 hello.i，hello.i 文件内容如下：

```
# 1 "hello.c"
```

```
# 1 "<built-in>"
# 1 "<command-line>"
# 1 "hello.c"
……
extern int printf (const char *__restrict __format, ...);
……
int main()
{
    printf("Hello World!");
    return 0;
}
```

比如文件包含语句#include<stdio.h>，预编译器会将 stdio.h 的文件内容拷贝到#include 语句声明的位置。如果源文件内使用#define 语句定义了宏，预编译器则将该宏的内容替换到其被引用的位置。如果宏定义本身使用了其他宏，则预编译器需要将宏递归地展开。

我们可以将预编译的工作简单地理解为源码的文本替换，即将宏定义的内容替换到宏的引用位置。当然，这样理解有一定的片面性，因为要考虑宏定义中使用其他宏的情况。事实上预编译器的实现机制和编译器有着很大的相似性，因此本书描述的编译系统将重点放在源代码的编译上，不再独立实现预编译器。然而，我们需要清楚的事实是：一个完善的编译器是需要预编译器的。

1.3.2　编译

接下来 GCC 对 hello.i 进行编译，命令如下：

```
$gcc -S hello.i -o hello.s
```

编译后产生的汇编文件 hello.s 内容如下：

```
    .file           "hello.c"
    .section        .rodata
.LC0:
    .string     "Hello World!"
    .text
.globl main
    .type       main, @function
main:
    pushl           %ebp
    movl            %esp, %ebp
    andl            $-16, %esp
    subl            $16, %esp
    movl            $.LC0, %eax
    movl            %eax, (%esp)
    call            printf
    movl            $0, %eax
    leave
    ret
    .size       main, .-main
```

试读结束：需要全本请在线购买： www.ertongbook.com