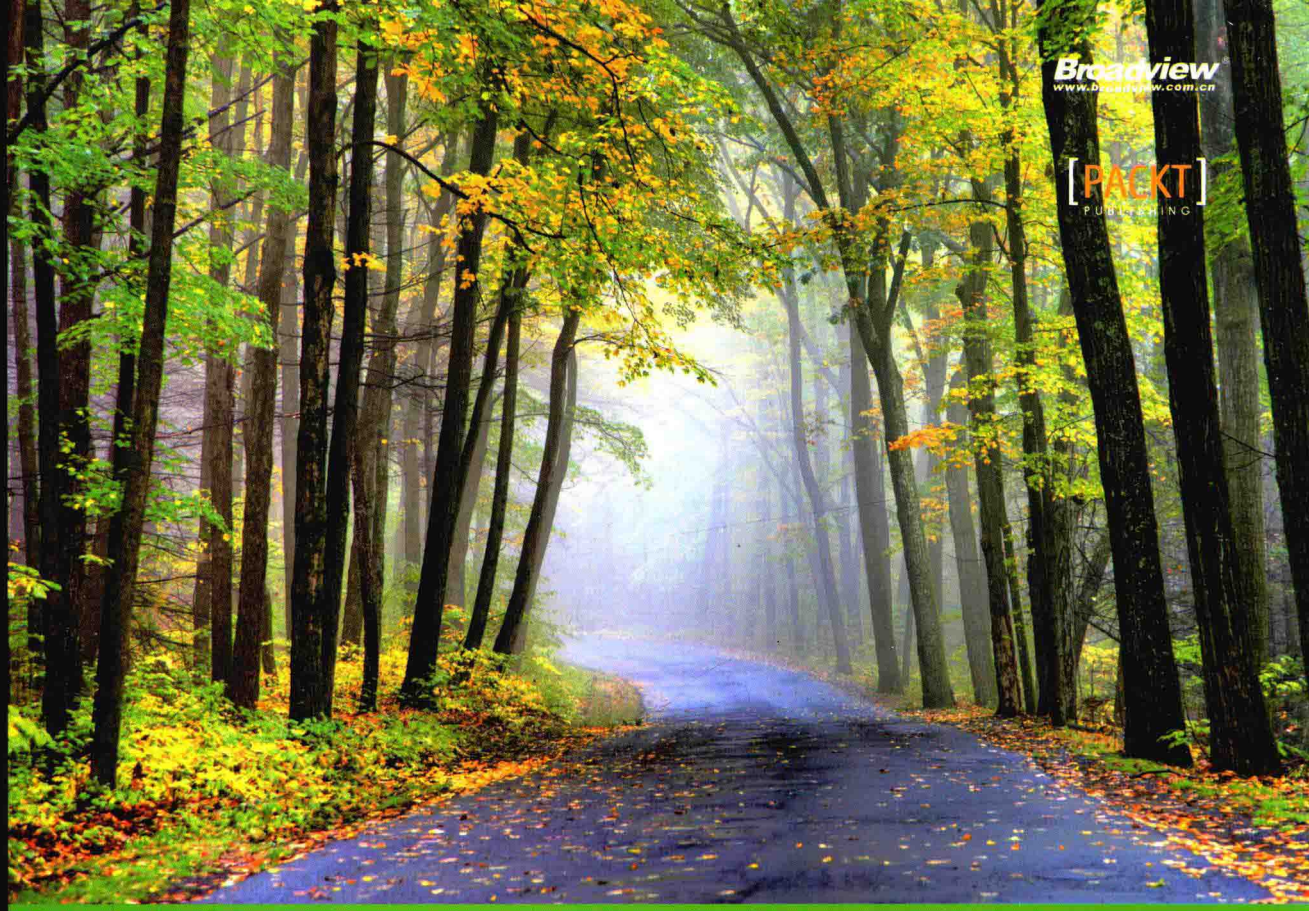


LLVM Cookbook

中文版

针对常见问题的快速解答
帮助你构建基于LLVM的编译器前端、优化器和代码生成器

【印】Mayur Pandey Suyog Sarda 著
王欢明 译

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

LLVM Cookbook

LLVM Cookbook 中文版

【印】Mayur Pandey Suyog Sarda 著
王欢明 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书以任务驱动的方式，带领读者编写基于 LLVM 的编译器前端、优化器、后端。通过丰富的实例，读者能够从中理解 LLVM 的架构，以及如何使用 LLVM 来编写自己的编译器。

相比于传统的介绍编译技术的书籍，此书更偏向于实战，因此适合熟悉编译但对 LLVM 比较陌生的人员，也适合正在学习编译技术并且在寻找实战机会的人员。

版权贸易合同登记号　图字：01-2015-6374

图书在版编目（CIP）数据

LLVM Cookbook 中文版 /（印）潘迪（Pandey,M.），（印）撒达（Sarda,S.）著；王欢明译. —北京：电子工业出版社，2016.6

ISBN 978-7-121-28847-0

Ⅰ. ①L… Ⅱ. ①潘… ②撒… ③王… Ⅲ. ①编译程序－程序设计 Ⅳ. ①TP314

中国版本图书馆 CIP 数据核字(2016)第 108748 号

策划编辑：张春雨
责任编辑：付　睿
印　　刷：三河市双峰印刷装订有限公司
装　　订：三河市双峰印刷装订有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　邮编：100036
开　　本：787×980　1/16　印张：19.25　字数：375 千字
版　　次：2016 年 6 月第 1 版
印　　次：2016 年 6 月第 1 次印刷
定　　价：75.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819　faq@phei.com.cn。

译者序

LLVM 这个名字源于 Lower Level Virtual Machine，但这个项目并不局限于创建一个虚拟机，它已经发展成为当今炙手可热的编译器基础框架。LLVM 最初以 C/C++为编译目标，近年来经过众多机构和开源社区的努力，LLVM 已经能够为 ActionScript、D、Fortran、Haskell、Java、Objective-C、Swift、Python、Ruby、Rust、Scala 等众多语言提供编译支持，而一些新兴语言则直接采用了 LLVM 作为后端。可以说，LLVM 对编译器领域的发展起到了举足轻重的作用。

本书是目前为数不多的介绍 LLVM 的书籍。本书从 LLVM 的构建与安装开始说起，介绍了 LLVM 的设计思想、LLVM 工具链、前端、优化器、后端，涵盖了 LLVM 的绝大部分内容。本书以任务驱动的方式对内容进行介绍，围绕着实现 TOY 语言的编译器，每一章节都会带领读者编写代码。在第 2 章实现了编译器的前端，第 4、5 章逐步实现优化器，后面的章节则实现了编译器后端。书中以实践的方式进行讲述，既阐述了原理，又让读者参与到编译器的开发当中，这一方面降低了学习 LLVM 的门槛，另一方面也让读者在实践中理解 LLVM 的细节。

作为译者，我觉得能够翻译此书也是一种缘分。最初是因为一次偶然的机会，我接触了一些自然语言处理的内容，在此过程中我领悟了词法分析和语法分析是怎么一回事；之后凭借着自己先前了解的零零碎碎的知识，在没有系统学习过编译原理的情况下写出了自己的第一个解释器（当然它很不完备）；接着便去系统学习编译原理，由于有了一定的实践基础，理解那些概念也轻松了许多；而关于这本书的翻译，则是因为在豆瓣上看到了一位豆友转发的消息，遂联系出版社的张春雨老师；最后在翻译此书的过程中，也收获了很多。

所以在这里要感谢带我走近自然语言处理的那位朋友，要感谢转发此消息的那位豆友，还要感谢博文视点的张春雨老师。人生充满了机缘巧合，我很幸运能够遇见你们。

与此同时，我也希望此书能够揭开编译器的面纱，能够让国内更多的人了解编译技术。

王欢明

2015 年 8 月

关于作者

Mayur Pandey 是一名专业的软件工程师，同时也是一位开源软件的爱好者。他专注于编译器以及编译器工具的开发，是 LLVM 开源社区的活跃贡献者，也是 Tizen 编译器项目的一员，他对其他编译器也有着亲身实践经验。

Mayur 在印度阿拉哈巴德的 Motilal Nehru 国家技术研究所获得学士学位。目前居住在印度班加罗尔。

"我要感谢我的家人和朋友，是他们帮我料理其他事务并且鼓励我，才使得我能够完成这本书的创作。"

Suyog Sarda 是一名专业的软件工程师，同时也是一位开源软件的爱好者。他专注于编译器以及编译器工具的开发，是 LLVM 开源社区的活跃贡献者，也是 Tizen 编译器项目的一员。除此之外，Suyog 也参与了 ARM 和 x86 架构的代码改进工作。他对其他的编译器也有着亲身实践经验。他对编译器的主要研究在于代码优化和向量化。

除了编译器之外，Suyog 也对 Linux 内核的开发很感兴趣。他曾在 2012 年于迪拜由 Birla 技术协会举办的 IEEE 国际云计算技术应用大会的议程上发表技术论文，题为 "*Secure Co-resident Virtualization in Multicore Systems by VM Pinning and Page Coloring*"。他在印度普纳工程大学获得计算机学士学位。目前居住于印度班加罗尔。

"我要感谢我的家人和朋友，也向一直帮助我的 LLVM 开源社区致以谢意。"

关于审校者

Logan Chien 在台湾国立大学获得计算机科学硕士学位。他的研究方向包括编译器设计、编译器优化、虚拟机。他是一名全职的软件工程师，在空闲时间，他从事多个开源项目的开发工作，例如 LLVM、Android。Logan 在 2012 年加入 LLVM 项目的开发。

Michael Haidl 是一名高性能计算工程师，致力于多核架构的研究，例如 GPU、Intel Xeon Phi accelerator。他有着超过 14 年的 C++开发经验，在并行计算方面有着丰富经验，在多年的工作中开发出多种编程模型（CUDA）。他同时有计算机科学和物理学的学位。目前，Michael 在德国明斯德大学担任研究助理，一边写着他的 PhD 论文，一边致力于研究基于 LLVM 架构的 GPU 编译技术。

"我要感谢每天用微笑和爱来支持我的妻子，同时也向为 Clang/LLVM 和其他 LLVM 项目付出辛勤工作的整个 LLVM 社区致以谢意。正是有了他们，LLVM 项目才能茁壮成长。"

Dave（Jing）Tian 是弗罗里达大学计算机和信息工程学院的研究助理及 PhD 学生。他是 SENSEI 中心的创始人之一。他的研究方向包括系统安全、嵌入式系统安全、可信计算、安全的静态代码分析及向量化。他对 Linux 内核开发和编译器都有着浓厚的兴趣。

Dave 花了一年时间研究人工智能和机器学习，在俄勒冈州大学教过 Python 和操作系统。在此之前，他在阿尔卡特朗讯公司 Linux 控制平台开发组从事过 4 年时间的软件开发工作。他在中国获得学士学位，以及电气工程的硕士学位。你可以在 `root@davejingtian.org` 及 `http://davejingtian.org` 了解他。

"我要感谢这本书的作者，他做得很好。也感谢 Packt 出版社的编辑们，是他们润色了这本书并且给我审校这本书的机会。"

前言

程序员在编程时没有一刻可以离开编译器。简单来说，所谓编译器就是把人类可读的高级语言映射到机器执行码。但你知道这里面发生了什么吗？编译器在生成优化过的机器码之前还做了很多处理工作，一个好的编译器包含了很多复杂的算法。

这本书介绍了编译的几个阶段：前端处理、代码优化、代码生成等。为了将这个复杂的过程简化，LLVM 使用了模块化的思想，使得每一个编译阶段都被独立出来；LLVM 使用面向对象的 C++语言完成，为编译器开发人员提供了易用而丰富的编程接口和 API。所以，LLVM 可能是最容易学习的编译器框架了。

作为作者，我们认为简单的解决方案往往会比复杂的解决方案更加奏效；通过这本书，我们将会了解许多编译技术，它能提升你的能力，让你了解编译选项，理解编译过程。

我们也相信，那些从事编译器开发的程序员会从本书收益良多，因为对编译器技术的了解会帮助他们写出更好的代码。

我们希望你能喜欢这本书，享受这本书提供的技术盛宴，也能开发自己的编译器。迫不及待了吗？让我们开始吧。

本书概述

第 1 章：LLVM 设计与使用。本章介绍了模块化的 LLVM 基础架构设计，让你学会如何下载安装 LLVM 和 Clang，通过一些例子来了解如何使用 LLVM 工作，也会介绍一些其他的编译器前端。

第 2 章：实现编译器前端。本章介绍了如何为一门编程语言编写一个编译器前端，我们通过为一门玩具语言写一个玩具编译器，来了解如何把前端语言映射到 LLVM IR。

第 3 章：扩展前端并增加 JIT 支持。本章为这门玩具语言增加了一些现代语言的高级

特性，以及对前端的 JIT 支持。

第 4 章：准备优化。本章介绍 LLVM IR 的 Pass 结构，以及不同的优化级别和每一级别上的优化技术。我们也将看到如何一步一步编写自己的 LLVM Pass。

第 5 章：实现优化。本章介绍如何在 LLVM IR 上实施诸多优化 Pass，以及在 LLVM 开源代码上实现一些向量化技术。

第 6 章：平台无关代码生成器。本章介绍了一个平台无关代码生成器的抽象结构，如何把 LLVM IR 转换到有向无环图（DAG），以及如何进一步生成目标平台机器码。

第 7 章：机器码优化。本章介绍了 DAG 的优化过程，目标寄存器分配算法，还介绍了 Selection DAG 上的各种优化技术及不同寄存器的分配技术。

第 8 章：实现 LLVM 后端。本章介绍了目标架构，包括寄存器、指令集、调用约定、编码、子平台特性等。

第 9 章：LLVM 项目最佳实践。本章介绍了一些使用 LLVM IR 做代码分析的其他项目。需要记住的是，LLVM 不仅仅是一个编译器，而且是一个编译器框架。本章介绍了一段可应用到各种项目的代码，可从中获取有用信息。

阅读背景

你只需要一台 Linux 计算机，最好是 Ubuntu 系统，就能完成本书的大部分例子。你也需要一个简单的文本或代码编辑器、网络连接，以及一个浏览器。我们建议安装两个文件的合并包，它在大部分 Linux 平台都能运行。

读者对象

本书适合那些熟悉编译器概念并且想理解学习 LLVM 的程序员。

本书也适合不直接参与编译器开发但参与大量代码开发的程序员。具备一定的编译器知识将会使你写出更加优秀的代码。

内容组织

在此书中你会频繁地看到一些标题，例如准备工作、详细步骤、工作原理、更多内容、另请参阅。

为了更好地呈现本书内容，我们采用了如下的组织方式。

准备工作

这部分对章节做了概述，并且描述了如何配置软件及其他工具。

详细步骤

这部分涵盖了具体的实践步骤。

工作原理

这部分涵盖了前一部分的详细解释。

更多内容

这部分涵盖了关于章节的更多信息。

另请参阅

这部分涵盖了参考资料的链接。

约定

在本书中你会发现大量用不同格式展示的文字，这里举例说明它们的涵义。

嵌入代码、数据库表名、目录名、文件名、文件扩展名、路径名、URL、用户输入、Twitter 用如下方式展示：“我们可以用 `include` 指令引入其他的上下文。”

代码块用如下格式：

```
primary := identifier_expr :
```

```
=numeric_expr
:=paran_expr
```

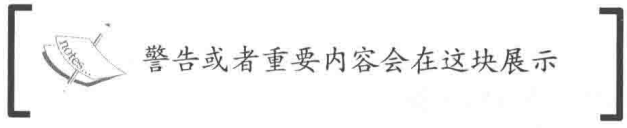
当我们想强调部分代码块时，相关行会使用粗体：

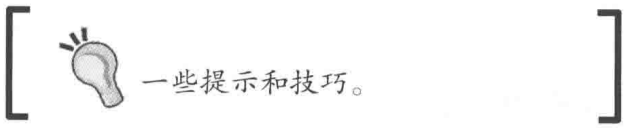
```
primary := identifier_expr
:=numeric_expr
:=paran_expr
```

命令行输入和输出用如下格式：

```
$ cat testfile.ll
```

新的术语和**重要单词**也会用黑体显示。你在屏幕上看到的内容，包括对话框或菜单，会这样显示：“单击**下一步**将进入下一屏”。





下载示例代码

你可以从 http://www.broadview.com.cn 下载所有已购买的博文视点书籍的示例代码文件。

勘误表

虽然我们已经尽力谨慎地确保内容的准确性，但错误仍然存在。如果你发现了书中的错误，包括正文和代码中的错误，请告诉我们，我们会非常感激。这样，你不仅帮助了其他读者，也帮助我们改进后续的出版。如发现任何勘误，可以在博文视点网站相应图书的页面提交勘误信息。一旦你找到的错误被证实，你提交的信息就会被接受，我们的网站也会发布这些勘误信息。你可以随时浏览图书页面，查看已发布的勘误信息。

目录

第 1 章
LLVM 设计与使用

本章涵盖以下话题。

- 模块化设计
- 交叉编译 Clang/LLVM
- 将 C 源码转换为 LLVM 汇编码
- 将 LLVM IR 转换为 bitcode
- 将 LLVM bitcode 转换为目标平台汇编码
- 将 LLVM bitcode 转回为 LLVM 汇编码
- 转换 LLVM IR
- 链接 LLVM bitcode
- 执行 LLVM bitcode
- 使用 C 语言前端——Clang
- 使用 GO 语言前端
- 使用 DragonEgg

概述

本节介绍 **LLVM** 的设计理念，以及如何使用 LLVM 提供的诸多工具。你将了解如何把 C 语言代码编译为 LLVM IR（Intermediate Representation——中间码）以及如何把它转为其他多种形式。你也会看到在 LLVM 的源码树中代码是如何组织的，以及如何使用 LLVM 自己编写一个编译器。

模块化设计

与其他编译器（例如 **GNU Compiler Collection——GCC**）不同，LLVM 的设计目标是成为一系列的库。本节以 LLVM 优化器（optimizer）为例来解释这个概念，因为它的设计就是基于库的。它允许你选择各个 Pass（趟）的执行顺序，也能够选择执行哪些优化 Pass——也就是说，有一些优化对你设计的系统是没有帮助的，只有少数优化会针对你的系统。反观传统的编译器优化器，它们通常是由大量高度耦合的代码组成，很难拆分成容易理解和使用的小模块。而在 LLVM 中，如果你想了解特定的优化器，是不需要知道整个系统是如何工作的。你只需选择一个优化器并使用它，无须担心其他依赖它的组件。

在我们开始本节之前，我们需要知道一点关于 LLVM 汇编码的知识。LLVM 的代码有 3 种表示形式：内存编译器中的 **IR**、存于磁盘的 bitcode，以及用户可读的汇编码。LLVM IR 是基于**静态单赋值**[1]（Static Single Assignment——SSA）的，并且提供了类型安全性、底层操作性、灵活性，因此能够清楚表达绝大多数高级语言。这种表示形式贯穿 LLVM 编译的各个阶段。事实上，LLVM IR 致力于成为一种足够底层的通用 IR，只有这样，高级语言的诸多特性才能够得以实现。同样，LLVM IR 组织良好，也具备不错的可读性。如果你对理解本节提到的 LLVM 汇编码有任何疑问，请参考本节结尾的“另请参阅”一节。

SSA 于 1980 年由 IBM 开始研究，由于它的一些良好性质，之后在编译器领域得到广泛应用，包括 LLVM。

准备工作

在开始之前，我们需要在本机安装 LLVM 工具链，特别是 `opt` 工具。

详细步骤

我们将在同一段代码上逐步实施两个不同的优化，来观察它们分别是如何改变代码的。

1. 首先，我们来写一段代码用作优化器的输入，在这里创建 `testfile.ll` 文件。

```
$ cat testfile.ll
define i32 @test1(i32 %A) {
  %B = add i32 %A, 0
```

1 在编译器的设计中，静态单赋值形式是一种特殊形式的中间码——每个变量仅被赋值一次。——译者注

```
  ret i32 %B
}

define internal i32 @test(i32 %X, i32 %dead) {
  ret i32 %X
}

define i32 @caller() {
  %A  = call i32 @test(i32 123, i32 456)
  ret i32 %A
}
```

2. 现在，使用 opt 工具来进行一个优化——指令合并。

```
$ opt -S -instcombine testfile.ll -o output1.ll
```

3. 查看输出，看看 instcombine 优化是如何进行的：

```
$ cat output1.ll
; ModuleID = 'testfile.ll'

define i32 @test1(i32 %A) {
  ret i32 %A
}

define internal i32 @test(i32 %X, i32 %dead) {
  ret i32 %X
}

define i32 @caller() {
  %A = call i32 @test(i32 123, i32 456)
  ret i32 %A
}
```

4. 使用 opt 工具进行无用参数消除（dead-argument-elimination）优化：

```
$ opt -S -deadargelim testfile.ll -o output2.ll
```

5. 查看输出，看看 deadargelim 优化的效果如何：