

LLVM 教程翻译  
Chinese Edition 纯中文版

alan2lin

May 2021



# Contents

<b>1 LLVM 系统入门</b>	<b>1</b>
1.1 概述 . . . . .	1
1.2 获取源代码并构建 LLVM . . . . .	1
1.3 要求 . . . . .	3
1.4 硬件 . . . . .	3
<b>2 The Second Chapter</b>	<b>5</b>



# Chapter 1

## LLVM 系统入门

### 1.1 概述

欢迎来到 LLVM 项目!

LLVM 项目包含多个组件。该项目的核心本身称为“LLVM”。它包含处理中间表示层并将其转换为目标文件所需的所有工具，库和头文件。工具包括汇编器，反汇编器，位码<sup>1</sup>分析器和位码优化器。它还包含基本的回归测试。

类 C 语言使用 Clang 前端。该组件将 C, C++, Objective C 和 Objective C++ 代码编译为 LLVM 位码，并使用 LLVM 将位码编译为目标文件。

其他组件包括：libc++ C++ 标准库，LLD 链接器等。

### 1.2 获取源代码并构建 LLVM

LLVM 入门文档可能已过时。Clang 入门页可能包含更准确的信息。这是一个获取和构建 llvm 源码的工作流程和配置的例子：

1. 检出 LLVM（包括相关的子项目，如 Clang）：

- `git clone https://github.com/llvm/llvm-project.git`
- 或者在 windows 上 `git clone -config core.autocrlf=false https://github.com/llvm/llvm-project.git`
- 为了节省存储和加快检出时间，你可能想要一个浅拷贝。例如，要获取 LLVM 项目的最后一次修订<sup>2</sup>，使用 `git clone -depth 1 https://github.com/llvm/llvm-project.git`

---

<sup>1</sup>llvm 里的中间表示层，跟 java 的 bytecode 一样

<sup>2</sup>在 git 或者 svn 等版本管理软件中，一次提交就会形成一次修订

## 2. 配置并构建 LLVM 和 Clang:

- cd llvm-project
- mkdir build
- cd build
- cmake -G < 生成器 > [选项] ../llvm

一些常见的构建系统生成器:

- Ninja —用于生成 Ninja <sup>3</sup> 构建文件. 大多数的 llvm 开发者使用 Ninja。
- Unix Makefiles —用于生成与 make 兼容的并行编译 makefiles。
- Visual Studio —用于生成 Visual Studio 项目和解决方案。
- Xcode —用于生成 Xcode 项目。

一些常见的选项:

- -DLLVM\_ENABLE\_PROJECTS='...' 你要额外构建的 LLVM 子项目的分号分隔列表, 可以包含以下任意项: clang, clang-tools-extra, libcxx, libcxxabi, libunwind, lldb, compiler-rt, lld, polly, 或 debuginfo-tests。  
例如要构建 LLVM, Clang, libcxx 和 libcxxabi, 使用 -DLLVM\_ENABLE\_PROJECTS="clang;clang-tools-extra;libcxx;libcxxabi;lld;polly;compiler-rt;debuginfo-tests"
- -DCMAKE\_INSTALL\_PREFIX= 目录—为目录指定要安装 LLVM 工具和库的位置的完整路径名 (默认为 /usr/local)。
- -DCMAKE\_BUILD\_TYPE= 类型—类型的有效选项为 Debug、Release、RelWithDebInfo 和 MinSizeRel。默认为 Debug。
- -DLLVM\_ENABLE\_ASSERTIONS= 开—启用断言检查进行编译 (调试版本的默认值为 Yes, 所有其他版本类型的默认值为 No)。Compile with assertion checks enabled (default is Yes for Debug builds, No for all other build types)。
- cmake -build . [-target < 目标 >] 或直接上面所指定的构建系统。
  - 默认目标 (例如: cmake -build . 或 make) 将构建所有的 LLVM 内容。
  - check-all 目标 (例如 ninja check-all) 将运行回归测试以确保一切正常。

---

<sup>3</sup>Ninja 是一个专注于速度的小型构建系统, 类似 make 但比 make 快

- CMake 会为每个工具和库生成构建目标，大多数 LLVM 子项目会生成自己的 `check-<project>` 目标
- 运行串行构建会很慢。要提高速度，请尝试运行并行构建。这是在 Ninja 中默认完成的；对于 make，使用选项 `-j NN`，其中 NN 是并行作业的数量，例如可用 CPU 的数量。
- 更多信息参见 CMake。
- 如果你遇到”内部编译错误”或者测试失败，参见下文。

欲知入门指南关于配置和编译 LLVM 详细信息，请转到目录布局去了解源码树的布局

## 1.3 要求

在开始使用 LLVM 系统之前，请查看下面给出的要求。通过提前了解您需要哪些硬件和软件，这可能会为您省去一些麻烦。

## 1.4 硬件

已经知道 LLVM 可以在以下主机平台运行:

OS	Arch	Compilers
Linux	x86 <sup>1</sup>	GCC, Clang
Linux	amd64	GCC, Clang
Linux	ARM	GCC, Clang
Linux	Mips	GCC, Clang
Linux	PowerPC	GCC, Clang
Linux	SystemZ	GCC, Clang
Solaris	V9 (Ultrasparc)	GCC
FreeBSD	x86 <sup>1</sup>	GCC, Clang
FreeBSD	amd64	GCC, Clang
NetBSD	x86 <sup>1</sup>	GCC, Clang
NetBSD	amd64	GCC, Clang
macOS2	PowerPC	GCC
macOS	x86	GCC, Clang
Cygwin/Win32	x86 <sup>1,3</sup>	GCC
Windows	x86 <sup>1</sup>	Visual Studio
Windows x64	x86-64	Visual Studio





## Chapter 2

### The Second Chapter

aaa Start to End aaaaaaaaaaaaaa S2E aaaaaaaaa 术语

# 词汇表

A      A is ...

B      B is ...

C      C is ...

LaTeX LaTeX is ...

中文测试



# 缩略语表

**S2E** Start to End. 6



# 术语表

**位码** llvm 里的中间表示层, 跟 java 的 bytecode 一样. 1

**忍者** Ninja 是一个专注于速度的小型构建系统, 类似 make 但比 make 快. 2

**术语** Acronyms and terms which are generally unknown or new to common readers.. 6

