LLVM 教程翻译

Bilingual Edition 双语版

alan2lin

May 2021

Contents

1 LLVM 系统入门 (Getting Started with the LLVM System) **1**

1.1 概述 (Overview) 1

2 The Second Chapter **5**

Chapter 1

LLVM 系统入门 (Getting Started with the LLVM System)

1.1 概述 (Overview)

欢迎来到 LLVM 项目!

Welcome to the LLVM project!

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[1]分析器和位码优化器。它还包含基本的回归测试。

The LLVM project has multiple components. The core of the project is itself called “LLVM”. This contains all of the tools, libraries, and header files needed to process intermediate representations and converts it into object files. Tools include an assembler, disassembler, bitcode analyzer, and bitcode optimizer. It also contains basic regression tests.

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。

C-like languages use the Clang front end. This component compiles C, C++, Objective C, and Objective C++ code into LLVM bitcode –and from there into object files, using LLVM.

其他组件包括：libc ++ C ++ 标准库，LLD 链接器等。

Other components include: the libc++ C++ standard library, the LLD linker, and more.

获取源代码并构建 LLVM

[1] llvm 里的中间表示层, 跟 java 的 bytecode 一样

Getting the Source Code and Building 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。

The LLVM Getting Started documentation may be out of date. The Clang Getting Started page might have more accurate information.

这是一个获取和构建 llvm 源码的工作流程和配置的例子:

This is an example workflow and configuration to get and build the LLVM source:

1. 检出 LLVM（包括相关的子项目，如 Clang):

   Checkout LLVM (including related subprojects like 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/l project.git

     or, on windows, 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[2], 使用 git clone –depth 1 https://github.com/llvm/llvm-project.git

     To save storage and speed-up the checkout time, you may want to do a shallow clone. For example, to get the latest revision of the LLVM project, use git clone –depth 1 https://github.com/llvm/llvm-project.git

2. 配置并构建 LLVM 和 Clang:

   Configure and build LLVM and Clang:

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

     cmake -G <generator> [options] ../llvm

     一些常见的构建系统生成器:

     Some common build system generators are:

     - Ninja —用于生成 Ninja [3] 构建文件. 大多数的 llvm 开发者使用 Ninja。

[2]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

[3]Ninja 是一个专注于速度的小型构建系统, 类似 make 但比 make 快

Ninja —for generating Ninja build files. Most llvm developers use Ninja.

- Unix Makefiles —用于生成与 make 兼容的并行编译 makeflies。
  Unix Makefiles —for generating make-compatible parallel makefiles.
- Visual Studio —用于生成 Visual Studio 项目和解决方案。
  Visual Studio —for generating Visual Studio projects and solutions.
- Xcode —用于生成 Xcode 项目。
  Xcode —for generating Xcode projects.

一些常见的选项:

Some Common options:

- -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。
  -DLLVM_ENABLE_PROJECTS='...' —semicolon-separated list of the LLVM subprojects you' d like to additionally build. Can include any of: clang, clang-tools-extra, libcxx, libcxxabi, libunwind, lldb, compiler-rt, lld, polly, or debuginfo-tests.
  例如要构建 LLVM,Clang,libcxx 和 libcxxabi, 使用 -DLLVM_ENABLE_PROJECTS=

  For example, to build LLVM, Clang, libcxx, and libcxxabi, use -DLLVM_ENABLE_PROJECTS="clang;libcxx;libcxxabi".
- -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_INSTALL_PREFIX=directory —Specify for directory the full pathname of where you want the LLVM tools and libraries to be installed (default /usr/local).
- -DCMAKE_BUILD_TYPE= 类型—类型的有效选项为 Debug、Release、RelWithDebInfo 和 MinSizeRel。默认为 Debug。
  -DCMAKE_BUILD_TYPE=type —Valid options for type are Debug, Release, RelWithDebInfo, and MinSizeRel. Default is 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).
-DLLVM_ENABLE_ASSERTIONS=On —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。
  cmake –build . [–target <target>] or the build system specified above directly.
  - 默认目标 (例如: cmake –build . 或 make) 将构建所有的 LLVM 内容。
    The default target (i.e. cmake –build . or make) will build all of LLVM.
  - check-all 目标（例如 ninja check-all）将运行回归测试以确保一切正常。
    The check-all target (i.e. ninja check-all) will run the regression tests to ensure everything is in working order.
  - 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
    CMake will generate build targets for each tool and library, and most LLVM sub-projects generate their own check-<project> target.
  - 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。
    Running a serial build will be slow. To improve speed, try running a parallel build. That' s done by default in Ninja; for make, use the option -j NN, where NN is the number of parallel jobs, e.g. the number of available CPUs.
- 更多信息参见 CMake。
  For more information see CMake
- 如果你遇到" 内部编译错误" 或者测试失败，参见下文。
  If you get an "internal compiler error (ICE)" or test failures, see below.

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

Consult the Getting Started with LLVM section for detailed information on configuring and compiling LLVM. Go to Directory Layout to learn about the layout of the source code tree.

Chapter 2

The Second Chapter

aaa Start to End aaaaaaaaaaaaaaa S2E aaaaaaaaaa 术语

词汇表

A A is ... 中文测试

B B is ...

C C is ...

LaTeX LaTeX is ...

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