Clang compiler frontend

Ivan Murashko

May 1, 2022

Contents

1	Introduction Environment setup														5								
2															7								
	2.1 Source code compilation															7							
		2.1.1	(Gei	ttir	ng t	he	sot	ırc	e c	ode	9											7
		2.1.2																					
		2.1.3	Ι	3u	ild			•															8
3	Arc	hitecti	ur	e																			9
4	Features																11						
	4.1	Preco	mp	pile	ed	hea	der	\mathbf{s}															11
		4.1.1	Ţ	Jse	er g	guid	de .																11
	4.2	Modu	ıles	3.																			12

4 CONTENTS

Introduction

The book describes clang compiler frontend internals. Architecture design is described and also practical code examples provided [2].

Environment setup

The chapter describes basic steps to be done to setup the environment be used for future experiments with clang.

2.1 Source code compilation

we are going to compile our source code in debug mode to be suitable for future investigations with debugger.

2.1.1 Getting the source code

The clang source is a part of LLVM. You can get it with the following command

```
git clone https://github.com/llvm/llvm-project.git
cd llvm-project
```

2.1.2 Configuration with cmake

Create a build folder where the compiler and related tools will be built

```
mkdir build cd build
```

Run configure script

```
cmake -G Ninja -DLLVM_ENABLE_PROJECTS="clang; clang-tools-extra" \
-DLLVM_USE_LINKER=gold -DLLVM_USE_SPLIT_DWARF=ON ../llvm
```

The are several options specified:

- LLVM_ENABLE_PROJECTS="clang; clang-tools-extra" specifies LLVM projects that we care about
- LLVM_USE_SPLIT_DWARF=ON spits debug information into separate files. This option saves disk space as well as memory consumption during the LLVM build. The option require compiler used for clang build to support it

2.1.3 Build

The build is trivial

ninja clang

You can also run unit and end-to-end tests for the compiler with

ninja check-clang

The compiler binary can be found as **bin/clang** at the build folder.

Architecture

You can find some info about clang internal architecture and relation with other LLVM components.

When we spoke about **clang** we should separate 2 things:

- driver
- compiler frontend

Both of them are called **clang** but perform different operations. The driver invokes different stages of compilation process (see fig. 3.1)

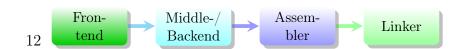


Figure 3.1: Clang driver

Features

You can find some info about different clang features at the chapter

4.1 Precompiled headers

Precompiled headers or **pch** is a clang feature that was designed with the goal to improve clang frontend performance. The basic idea was to create AST for a header file and reuse the AST for some purposes.

4.1.1 User guide

Generate you pch file is simple [1]. Suppose you have a header file with name header.h:

```
#pragma once
void foo() {
}
```

then you can generate a pch for it with

```
clang -x c++-header header.h -o header.pch
```

the option -x c++-header was used there. The option says that the header file has to be treated as a c++ header file. The output file is **header.pch**.

The precompiled headers generation is not enough and you may want to start using them. Typical C++ source file that uses the header may look like

```
// test pchs
#include "header.h"
int main() {
  foo();
  return 0;
   As you may see, the header is included as follows
#include "header.h"
By default clang will not use a pch at the case and you have to specify it
explicitly with
clang -include-pch header.pch main.cpp -o main -lstdc++
We can check the command with debugger and it will give us
$ lldb ~/local/llvm-project/build/bin/clang -- -cc1
\rightarrow -include-pch header.pch main.cpp -fsyntax-only
(lldb) b clang::ASTReader::ReadAST
. . .
(lldb) r
   4231
          llvm::SaveAndRestore<SourceLocation>
-> 4232
             SetCurImportLocRAII(CurrentImportLoc, ImportLoc);
   4233
          11vm::SaveAndRestore<Optional<ModuleKind>>

→ SetCurModuleKindRAII(
   4234
               CurrentDeserializingModuleKind, Type);
   4235
(lldb) p FileName
(llvm::StringRef) $0 = (Data = "header.pch", Length = 10)
Note that only the first -include-pch option will be processed, all others
```

Note that only the first **-include-pch** option will be processed, all others will be ignored. It reflects the fact that there can be only one precompiled header for a translation unit.

4.2 Modules

TBD

Bibliography

- [1] Clang compiler user's manual. 2022. https://clang.llvm.org/docs/UsersManual.html.
- [2] Murashko, I. Source code examples for clang compiler fronmtend.— 2022.— https://github.com/ivanmurashko/articles/clangbook/src.