LLVM

M了个J

https://github.com/CoderMJLee

https://weibo.com/exceptions

实力IT教育 www.520it.com

小码哥教育 SEEMYGO

LLVM

小码哥教育 SEEMYGO

LLVM

- 什么是LLVM
  - 官网：https://llvm.org/
  - The LLVM Project is a collection of modular and reusable compiler and toolchain technologies.
  - LLVM项目是模块化、可重用的编译器以及工具链技术的集合
  - 美国计算机协会 (ACM) 将其2012 年软件系统奖项颁给了LLVM，之前曾经获得此奖项的软件和技术包括：Java、Apache、Mosaic、the World Wide Web、Smalltalk、UNIX、Eclipse等等

- 创始人
  - Chris Lattner，亦是Swift之父

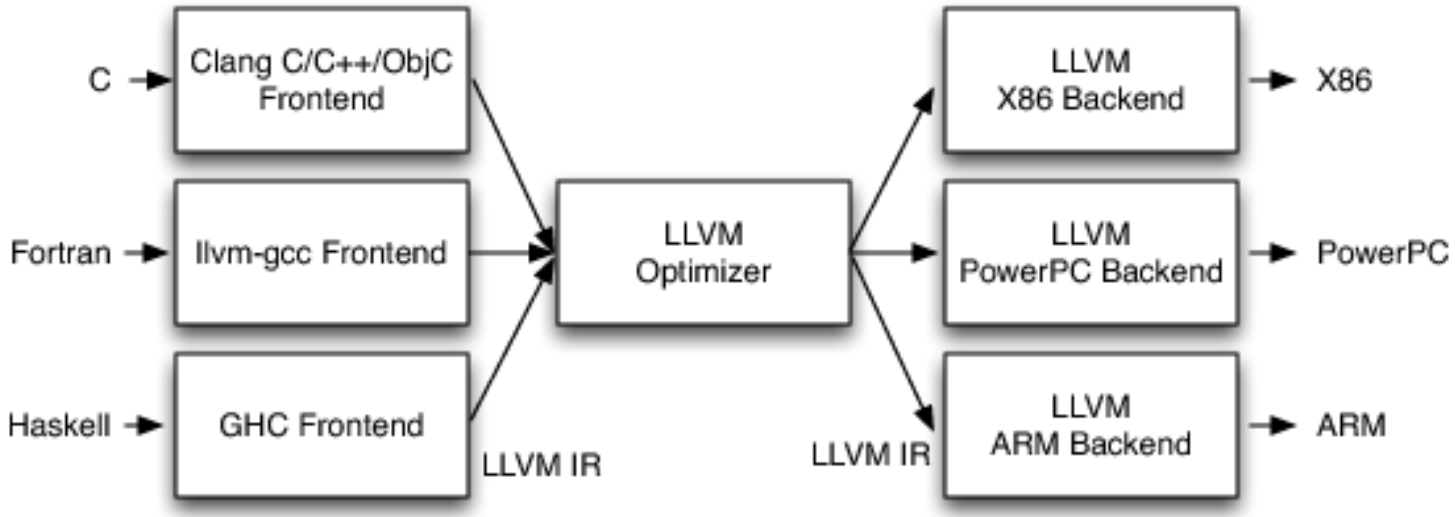
- 有些文章把LLVM当做Low Level Virtual Machine（低级虚拟机）的缩写简称，官方描述如下
  - The name "LLVM" itself is not an acronym; it is the full name of the project.
  - “LLVM”这个名称本身不是首字母缩略词; 它是项目的全名

小码哥教育 SEEMYGO

传统的编译器架构

- Frontend：前端
  - 词法分析、语法分析、语义分析、生成中间代码
- Optimizer：优化器
  - 中间代码优化
- Backend：后端
  - 生成机器码

LLVM架构



- 不同的前端后端使用统一的中间代码LLVM Intermediate Representation (LLVM IR)
- 如果需要支持一种新的编程语言，那么只需要实现一个新的前端
- 如果需要支持一种新的硬件设备，那么只需要实现一个新的后端
- 优化阶段是一个通用的阶段，它针对的是统一的LLVM IR，不论是支持新的编程语言，还是支持新的硬件设备，都不需要对优化阶段做修改
- 相比之下，GCC的前端和后端没分得太开，前端后端耦合在了一起。所以GCC为了支持一门新的语言，或者为了支持一个新的目标平台，就变得特别困难
- LLVM现在被作为实现各种静态和运行时编译语言的通用基础结构（GCC家族、Java、.NET、Python、Ruby、Scheme、Haskell、D等）

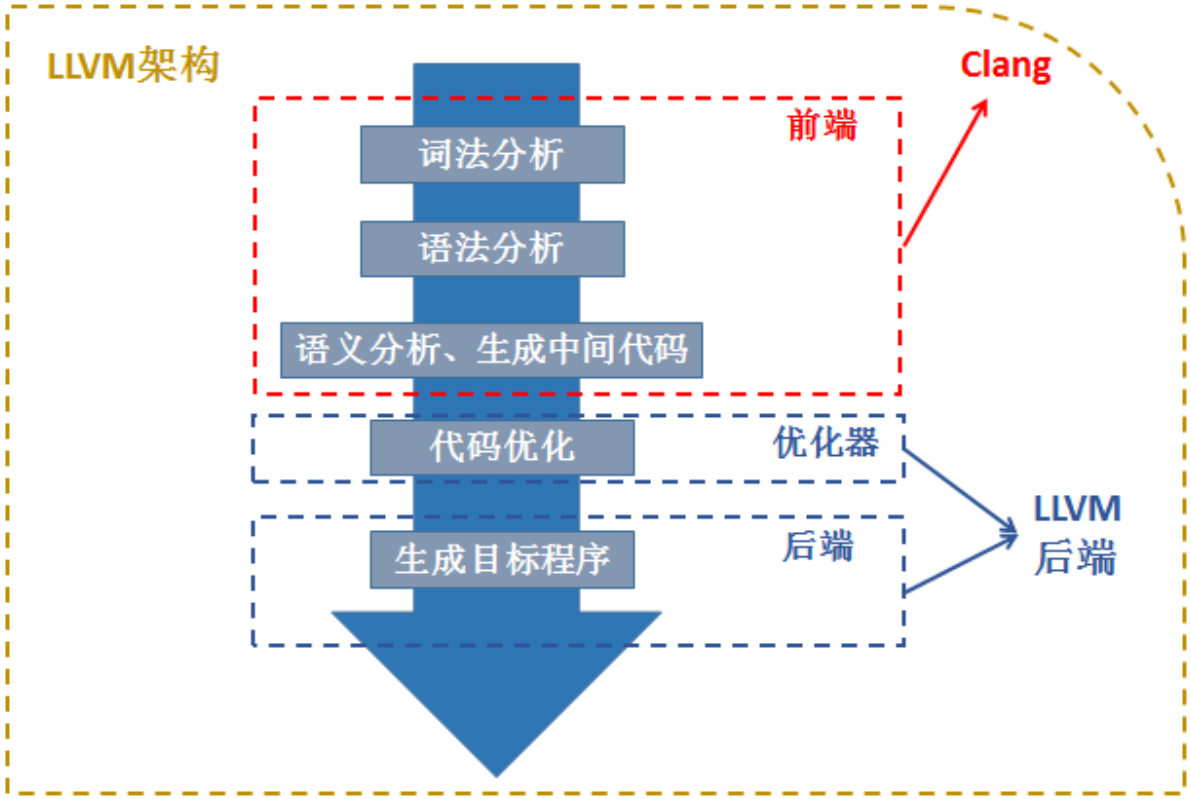
小码哥教育 SEEMYGO

Clang

- 什么是Clang？
  - LLVM项目的一个子项目
  - 基于LLVM架构的C/C++/Objective-C编译器前端
  - 官网：http://clang.llvm.org/

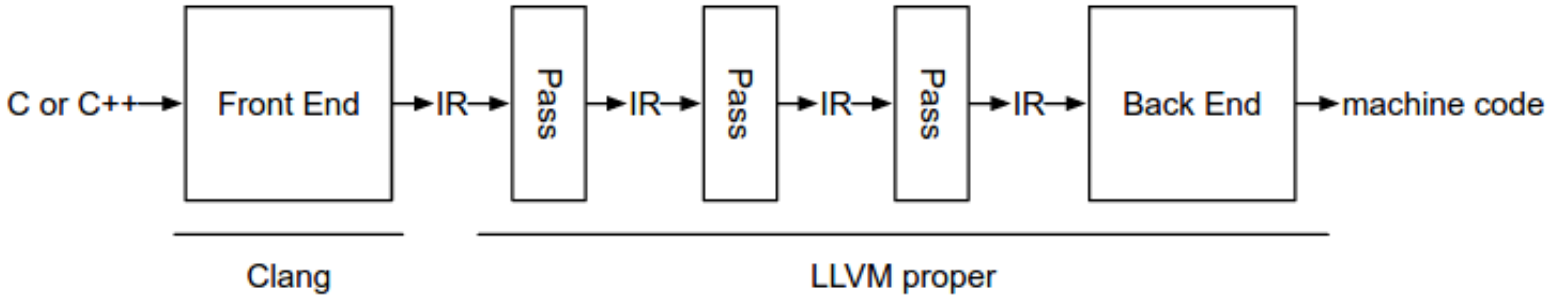
- 相比于GCC，Clang具有如下优点
  - 编译速度快：在某些平台上，Clang的编译速度显著的快过GCC（Debug模式下编译OC速度比GGC快3倍）
  - 占用内存小：Clang生成的AST所占用的内存是GCC的五分之一左右
  - 模块化设计：Clang采用基于库的模块化设计，易于 IDE 集成及其他用途的重用
  - 诊断信息可读性强：在编译过程中，Clang 创建并保留了大量详细的元数据 (metadata)，有利于调试和错误报告
  - 设计清晰简单，容易理解，易于扩展增强

Clang与LLVM



- 广义的LLVM
  - 整个LLVM架构

- 狭义的LLVM
  - LLVM后端（代码优化、目标代码生成等）



小码哥教育 SEEMYGO

OC源文件的编译过程

- 命令行查看编译的过程：$ clang -ccc-print-phases main.m

```
0: input, "main.m", objective-c
1: preprocessor, {0}, objective-c-cpp-output
2: compiler, {1}, ir
3: backend, {2}, assembler
4: assembler, {3}, object
5: linker, {4}, image
6: bind-arch, "x86_64", {5}, image
```

- 查看preprocessor（预处理）的结果：$ clang -E main.m

词法分析

- 词法分析，生成Token： $ clang -fmodules -E -Xclang -dump-tokens main.m

```
1
2 void test(int a, int b) {
3     int c = a + b - 3;
4 }
5
```

```
void 'void'      [StartOfLine]  Loc=<test.m:2:1>
identifier 'test'        [LeadingSpace] Loc=<test.m:2:6>
l_paren '('              Loc=<test.m:2:10>
int 'int'                Loc=<test.m:2:11>
identifier 'a'   [LeadingSpace] Loc=<test.m:2:15>
comma ','                Loc=<test.m:2:16>
int 'int'        [LeadingSpace] Loc=<test.m:2:18>
identifier 'b'   [LeadingSpace] Loc=<test.m:2:22>
r_paren ')'              Loc=<test.m:2:23>
l_brace '{'      [LeadingSpace] Loc=<test.m:2:25>
int 'int'        [StartOfLine] [LeadingSpace]   Loc=<test.m:3:5>
identifier 'c'   [LeadingSpace] Loc=<test.m:3:9>
equal '='        [LeadingSpace] Loc=<test.m:3:11>
identifier 'a'   [LeadingSpace] Loc=<test.m:3:13>
plus '+'         [LeadingSpace] Loc=<test.m:3:15>
identifier 'b'   [LeadingSpace] Loc=<test.m:3:17>
minus '-'        [LeadingSpace] Loc=<test.m:3:19>
numeric_constant '3'     [LeadingSpace] Loc=<test.m:3:21>
semi ';'                 Loc=<test.m:3:22>
r_brace '}'      [StartOfLine]  Loc=<test.m:4:1>
eof ''          Loc=<test.m:4:2>
```

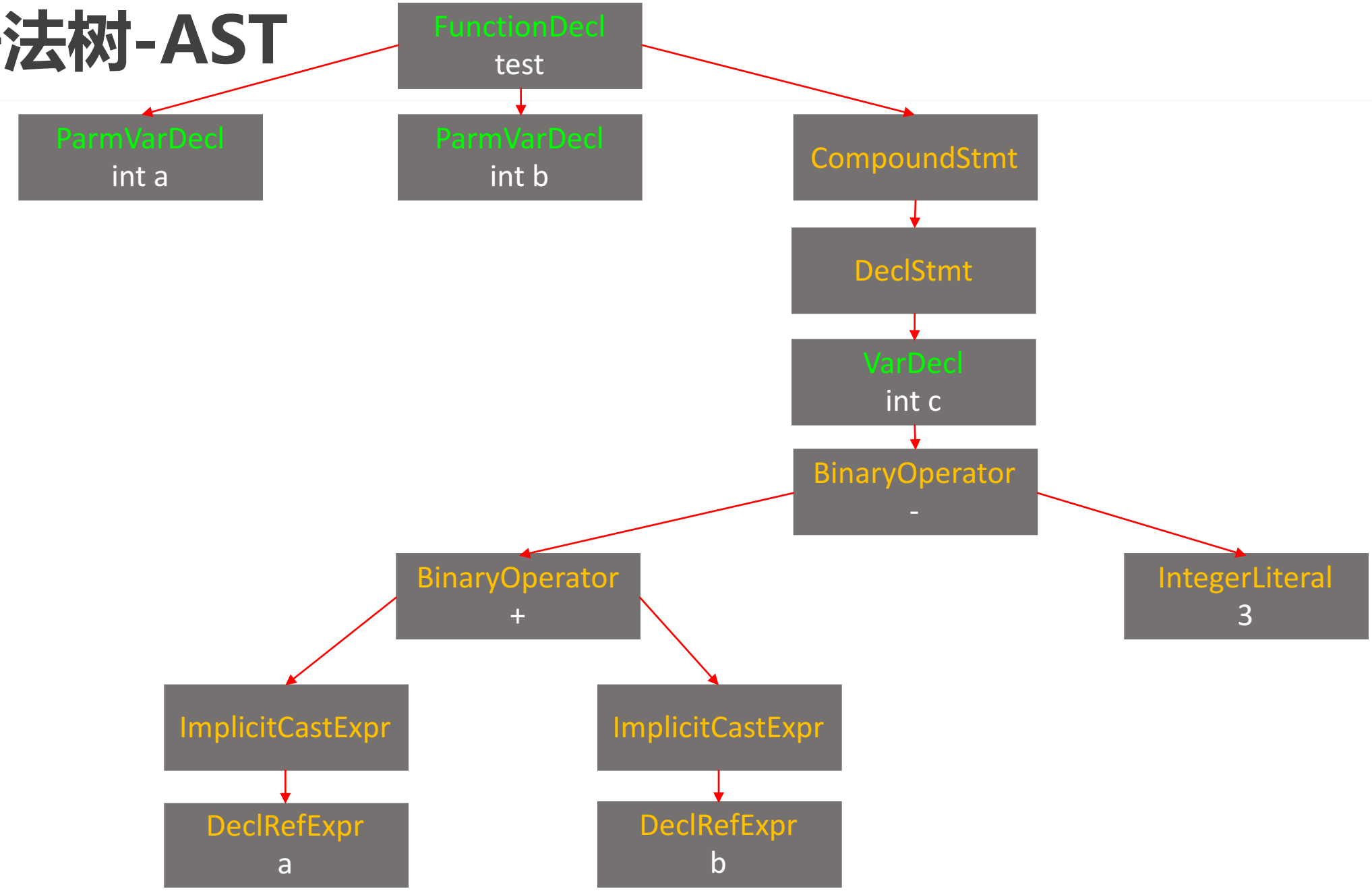
语法树-AST

- 语法分析，生成语法树（AST，Abstract Syntax Tree）：$ clang -fmodules -fsyntax-only -Xclang -ast-dump main.m

```
`-FunctionDecl 0x7fab75800870 <test.m:2:1, line:4:1> line:2:6 test 'void (int, int)'
  |-ParmVarDecl 0x7fab758006f8 <col:11, col:15> col:15 used a 'int'
  |-ParmVarDecl 0x7fab75800770 <col:18, col:22> col:22 used b 'int'
  `-CompoundStmt 0x7fab75800af0 <col:25, line:4:1>
    `-DeclStmt 0x7fab75800ad8 <line:3:5, col:22>
      `-VarDecl 0x7fab75800988 <col:5, col:21> col:9 c 'int' cinit
        `-BinaryOperator 0x7fab75800ab0 <col:13, col:21> 'int' '-'
          |-BinaryOperator 0x7fab75800a68 <col:13, col:17> 'int' '+'
          | |-ImplicitCastExpr 0x7fab75800a38 <col:13> 'int' <LValueToRValue>
          | | `-DeclRefExpr 0x7fab758009e8 <col:13> 'int' lvalue ParmVar 0x7fab758006f8 'a' 'int'
          | `-ImplicitCastExpr 0x7fab75800a50 <col:17> 'int' <LValueToRValue>
          |   `-DeclRefExpr 0x7fab75800a10 <col:17> 'int' lvalue ParmVar 0x7fab75800770 'b' 'int'
          `-IntegerLiteral 0x7fab75800a90 <col:21> 'int' 3
```

小码哥教育 SEEMYGO

语法树-AST



LLVM IR

- LLVM IR有3种表示形式（但本质是等价的，就好比水可以有气体、液体、固体3种形态）
  - text：便于阅读的文本格式，类似于汇编语言，拓展名.ll， $ clang -S -emit-llvm main.m
  - memory：内存格式
  - bitcode：二进制格式，拓展名.bc， $ clang -c -emit-llvm main.m

```
define void @test(i32, i32) #0 {
  %3 = alloca i32, align 4
  %4 = alloca i32, align 4
  %5 = alloca i32, align 4
  store i32 %0, i32* %3, align 4
  store i32 %1, i32* %4, align 4
  %6 = load i32, i32* %3, align 4
  %7 = load i32, i32* %4, align 4
  %8 = add nsw i32 %6, %7
  %9 = sub nsw i32 %8, 3
  store i32 %9, i32* %5, align 4
  ret void
}
```

- IR基本语法
  - 注释以分号；开头
  - 全局标识符以@开头，局部标识符以%开头
  - alloca，在当前函数栈帧中分配内存
  - i32，32bit，4个字节的意思
  - align，内存对齐
  - store，写入数据
  - load，读取数据

- 官方语法参考
  - https://llvm.org/docs/LangRef.html

小码哥教育 SEEMYGO

源码下载

- 下载LLVM
  - $ git clone https://git.llvm.org/git/llvm.git/
  - 大小 648.2 M，仅供参考

- 下载clang
  - $ cd llvm/tools
  - $ git clone https://git.llvm.org/git/clang.git/
  - 大小 240.6 M，仅供参考

源码编译

- 安装cmake和ninja（先安装brew，https://brew.sh/）
  - $ brew install cmake
  - $ brew install ninja

- ninja如果安装失败，可以直接从github获取release版放入【/usr/local/bin】中
  - https://github.com/ninja-build/ninja/releases

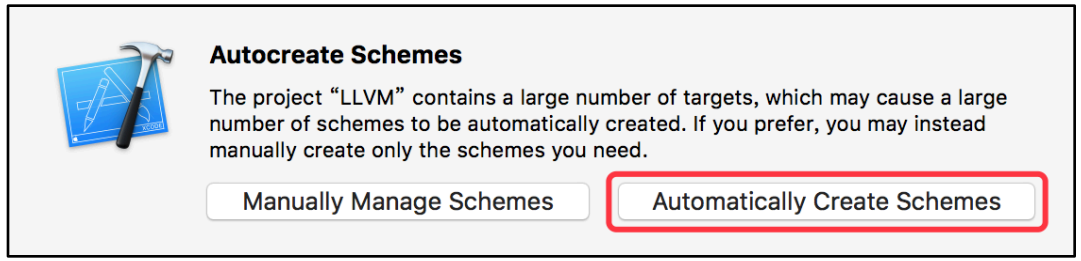
- 在LLVM源码同级目录下新建一个【llvm_build】目录（最终会在【llvm_build】目录下生成【build.ninja】）
  - $ cd llvm_build
  - $ cmake -G Ninja ../llvm -DCMAKE_INSTALL_PREFIX=LLVM的安装路径
  - 更多cmake相关选项，可以参考： https://llvm.org/docs/CMake.html

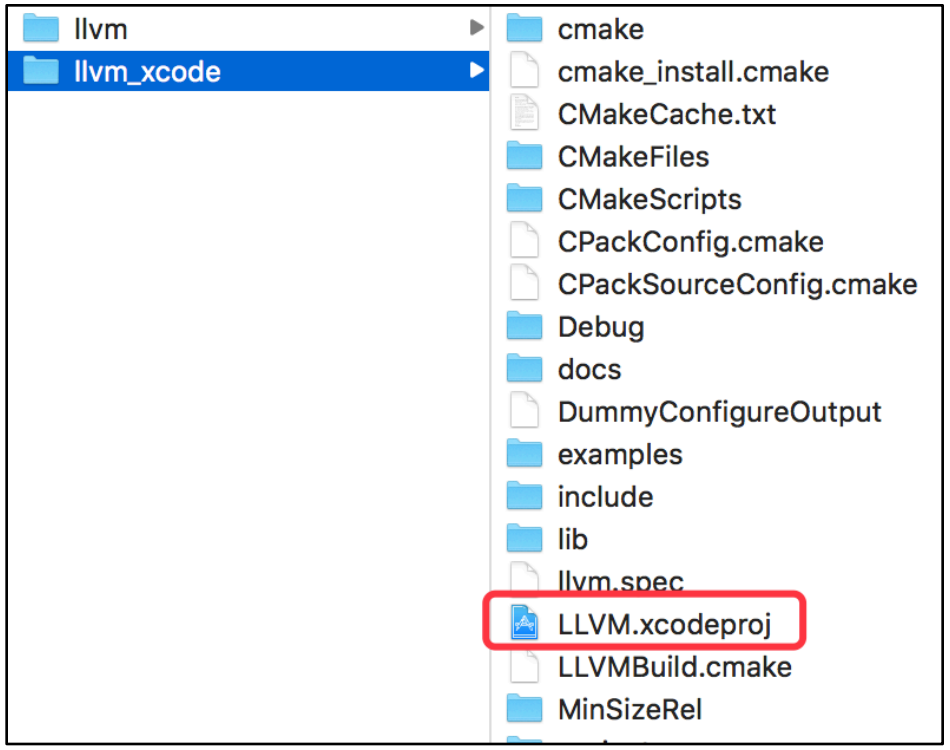
- 依次执行编译、安装指令
  - $ ninja
  - 编译完毕后，【llvm_build】目录大概 21.05 G（仅供参考）
  - $ ninja install
  - 安装完毕后，安装目录大概 11.92 G（仅供参考）

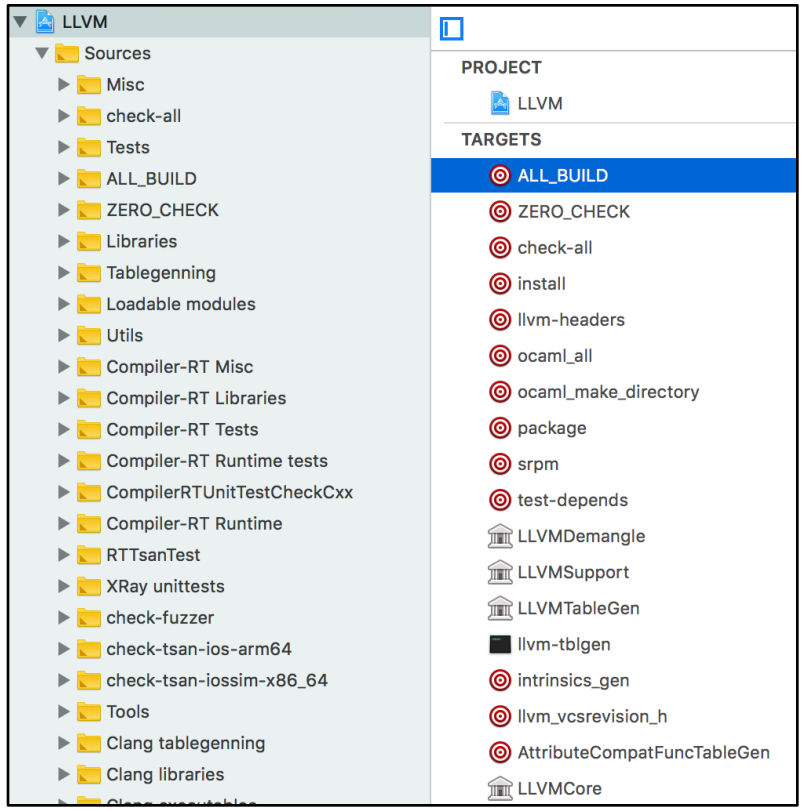
源码编译

- 也可以生成Xcode项目再进行编译，但是速度很慢（可能需要1个多小时）

- 在llvm同级目录下新建一个【llvm_xcode】目录
  - $ cd llvm_xcode
  - $ cmake -G Xcode ../llvm







小码哥教育 SEEMYGO

应用与实践

- libclang、libTooling
  - 官方参考：https://clang.llvm.org/docs/Tooling.html
  - 应用：语法树分析、语言转换等

- Clang插件开发
  - 官方参考
    - ✓ https://clang.llvm.org/docs/ClangPlugins.html
    - ✓ https://clang.llvm.org/docs/ExternalClangExamples.html
    - ✓ https://clang.llvm.org/docs/RAVFrontendAction.html
  - 应用：代码检查（命名规范、代码规范）等

小码哥教育 SEEMYGO

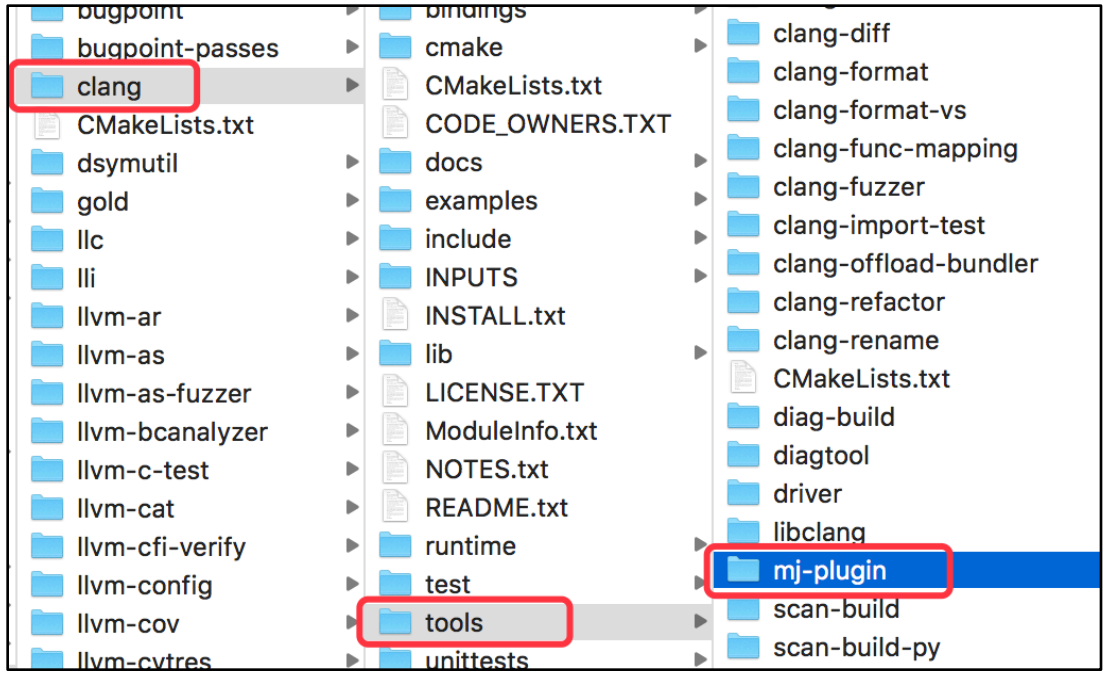
应用与实践

- Pass开发
  - 官方参考：https://llvm.org/docs/WritingAnLLVMPass.html
  - 应用：代码优化、代码混淆等

- 开发新的编程语言
  - https://llvm-tutorial-cn.readthedocs.io/en/latest/index.html
  - https://kaleidoscope-llvm-tutorial-zh-cn.readthedocs.io/zh_CN/latest/

clang插件开发1 – 插件目录

- 在【clang/tools】源码目录下新建一个插件目录，假设叫做【mj-plugin】



- 在【clang/tools/CMakeLists.txt】最后加入内容： add_clang_subdirectory(mj-plugin)，小括号里是插件目录名

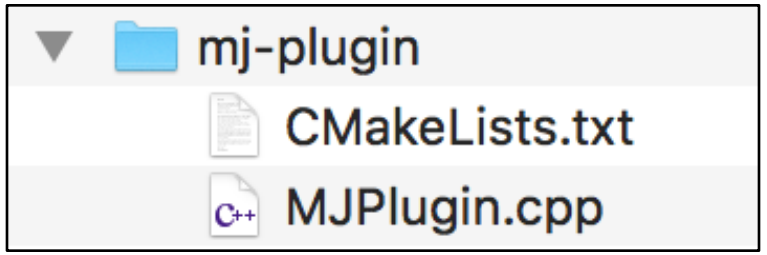
```
# libclang may require clang-tidy in
add_clang_subdirectory(libclang)
add_clang_subdirectory(mj-plugin)
```

clang插件开发2 – 插件必要文件

- 在【mj-plugin】目录下新建一个【CMakeLists.txt】，文件内容是：add_llvm_loadable_module(MJPlugin MJPlugin.cpp)
  - MJPlugin是插件名，MJPlugin.cpp是源代码文件

CMakeLists.txt

```
add_llvm_loadable_module(MJPlugin MJPlugin.cpp)
```



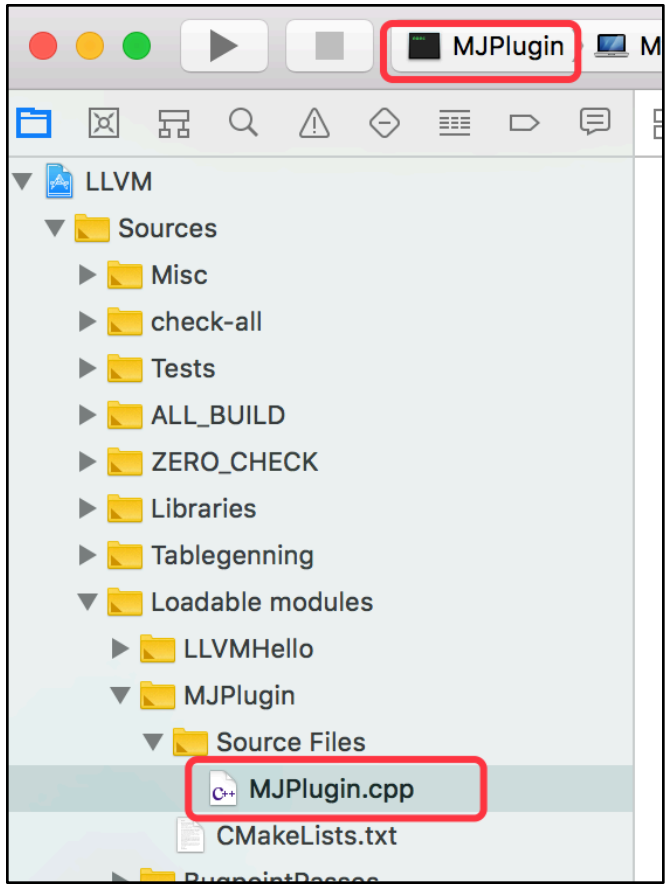
clang插件开发3 – 编写插件源码

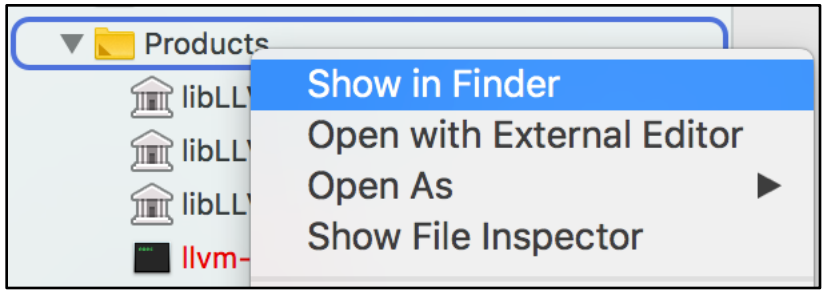
- 【MJPlugin.cpp】参考

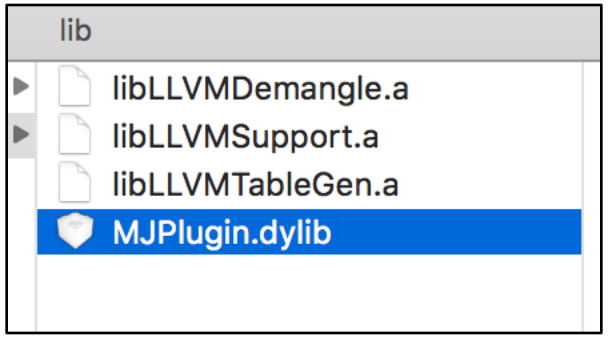
```cpp
#include <iostream>
#include "clang/AST/AST.h"
#include "clang/AST/ASTConsumer.h"
#include "clang/Frontend/CompilerInstance.h"
#include "clang/Frontend/FrontendPluginRegistry.h"

using namespace clang;
using namespace std;
using namespace llvm;

namespace MJPlugin {
    class MJASTConsumer: public ASTConsumer {
    public:
        void HandleTranslationUnit(ASTContext &context) {
            cout << "MJPlugin-HandleTranslationUnit" << endl;
        }
    };

    class MJASTAction: public PluginASTAction {
    public:
        unique_ptr<ASTConsumer> CreateASTConsumer(CompilerInstance &ci, StringRef iFile) {
            return unique_ptr<MJASTConsumer> (new MJASTConsumer);
        }

        bool ParseArgs(const CompilerInstance &ci, const vector<string> &args) {
            return true;
        }
    };
}

static FrontendPluginRegistry::Add<MJPlugin::MJASTAction>
X("MJPlugin", "The MJPlugin is my first clang-plugin.");
```

clang插件开发4 – 编译插件

- 利用cmake生成的Xcode项目来编译插件（第一次编写完插件，需要利用cmake重新生成一下Xcode项目）
- 插件源代码在【Sources/Loadable modules】目录下可以找到，这样就可以直接在Xcode里编写插件代码
- 选择MJPlugin这个target进行编译，编译完会生成一个动态库文件

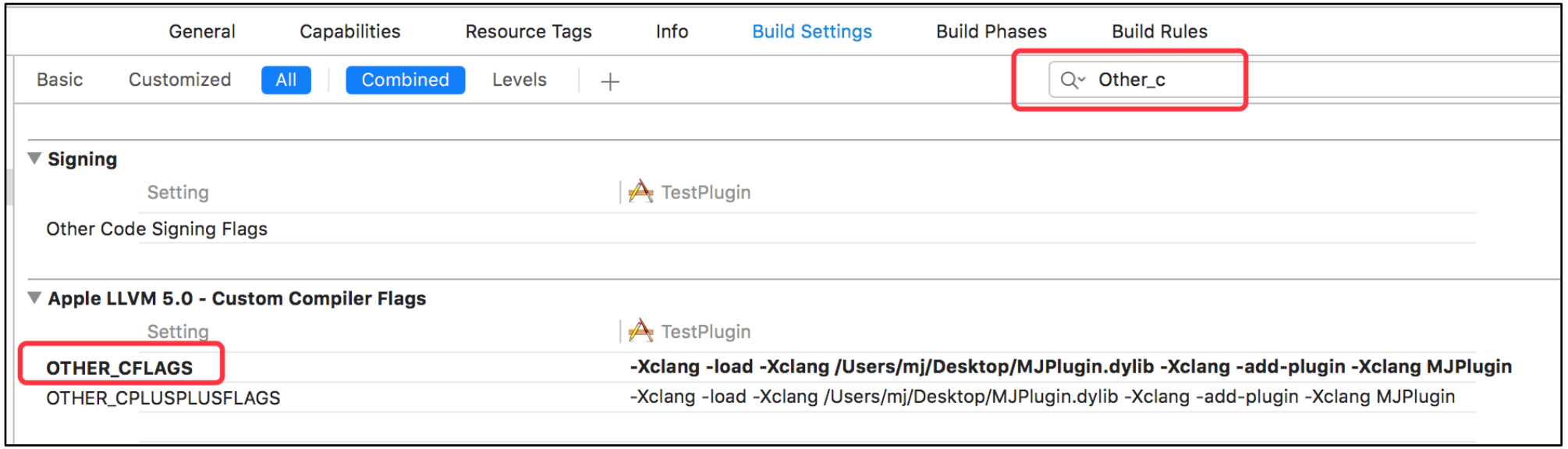






clang插件开发5 – 加载插件

- 在Xcode项目中指定加载插件动态库：Build Settings > OTHER_CFLAGS
  - -Xclang -load -Xclang 动态库路径 -Xclang -add-plugin -Xclang 插件名称



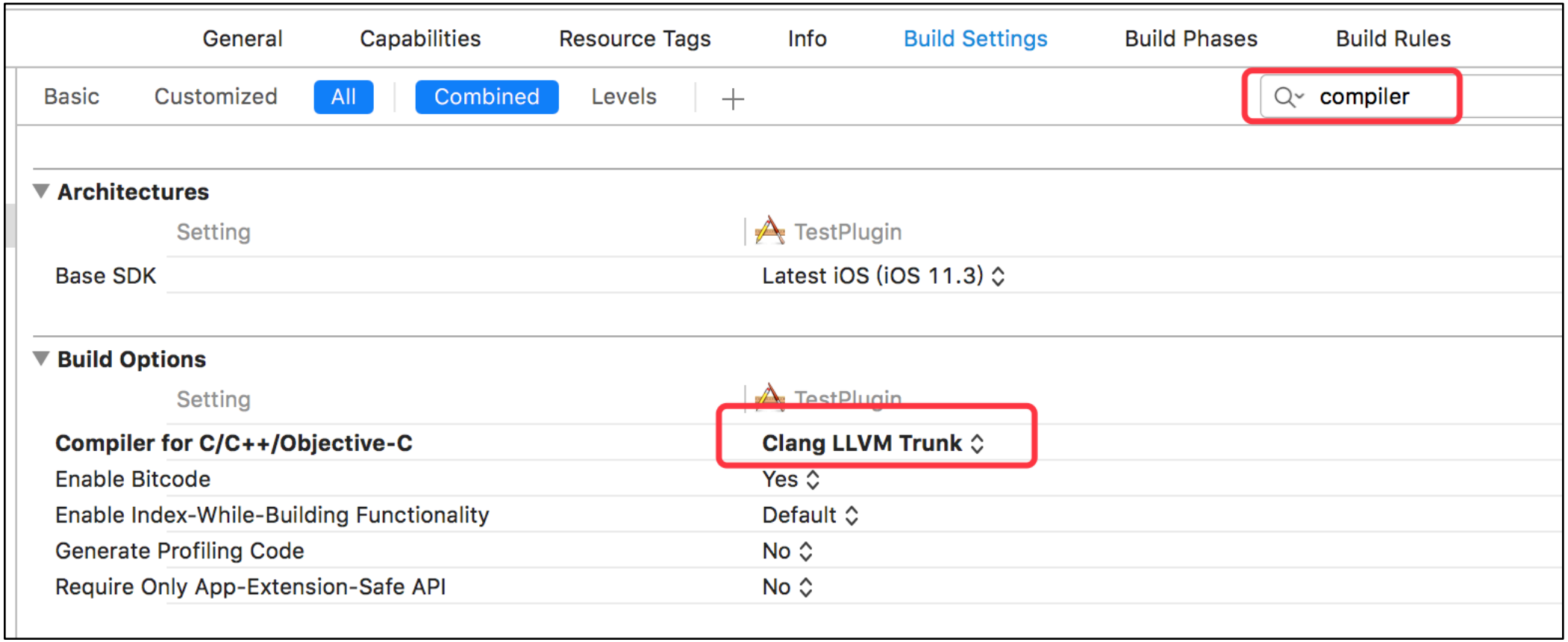
clang插件开发6 – Hack Xcode

- 首先要对Xcode进行Hack，才能修改默认的编译器
- 下载【XcodeHacking.zip】，解压，修改【HackedClang.xcplugin/Contents/Resources/HackedClang.xcspec】的内容，设置一下自己编译好的clang的路径

```
        sdk;
    );
    BuiltinJambaseRuleName = ProcessC;
    ExecPath = "/opt/llvm/llvm_build/bin/clang";
    UseCPlusPlusCompilerDriverWhenBundlizing = Yes;
    CommandOutputParser = "XCSimpleBufferedCommandOutputParser
    SupportsHeadermaps = Yes;
    DashIFlagAcceptsHeadermaps = Yes;
    SupportsIsysroot = Yes;
    SupportsPredictiveCompilation = No;
```

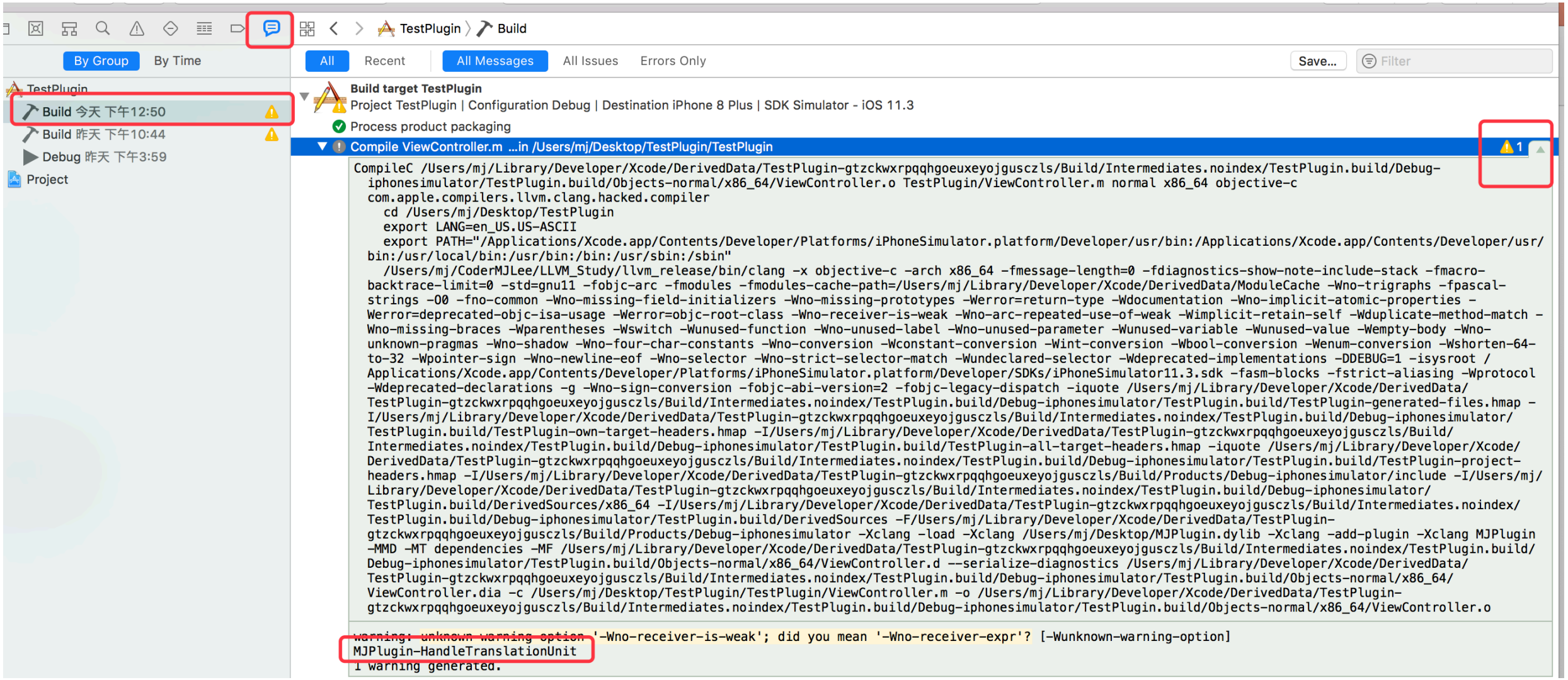
- 然后在XcodeHacking目录下进行命令行，将XcodeHacking的内容剪切到Xcode内部
- $ sudo mv HackedClang.xcplugin `xcode-select-print-path`/../PlugIns/Xcode3Core.ideplugin/Contents/SharedSupport/Developer/Library/Xcode/Plug-ins
- $ sudo mv HackedBuildSystem.xcspec `xcode-select-print-path`/Platforms/iPhoneSimulator.platform/Developer/Library/Xcode/Specifications

clang插件开发7 – 修改Xcode的编译器



clang插件开发8 – 编译项目

- 编译项目后，会在编译日志看到MJPlugin插件的打印信息（如果插件更新了，最好先Clean一下项目）



小码哥教育 SEEMYGO

clang插件开发9 – 更多

- 想要实现更复杂的插件功能，就需要利用clang的API针对语法树（AST）进行相应的分析和处理

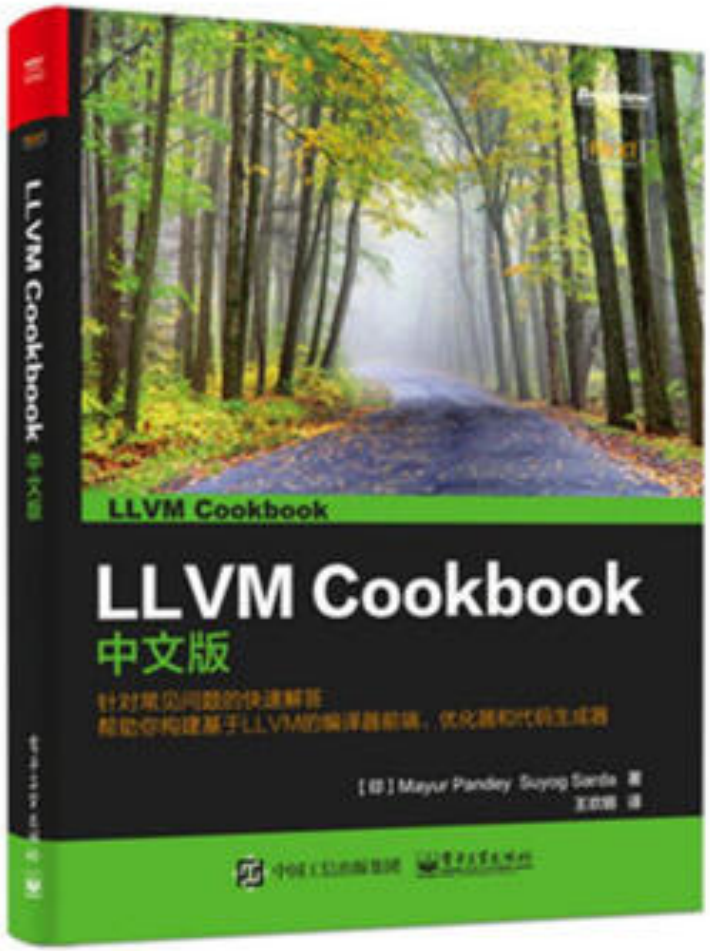
- 关于AST的资料
  - https://clang.llvm.org/doxygen/namespaceclang.html
  - https://clang.llvm.org/doxygen/classclang_1_1Decl.html
  - https://clang.llvm.org/doxygen/classclang_1_1Stmt.html

提示警告、错误信息

```
MJASTConsumer(CompilerInstance &ci) {
        DiagnosticsEngine &D = ci.getDiagnostics();
        D.Report(D.getCustomDiagID(DiagnosticsEngine::Warning, "MJPlugin警告信息"));
        D.Report(D.getCustomDiagID(DiagnosticsEngine::Error, "MJPlugin错误信息"));
}
```

小码哥教育
SEEMYGO

推荐书籍



小码哥教育 SEEMYGO

交流方式

- 昵称：码宝宝
- 微信号：IT-SEEMYGO

- 技术公众号
  - 各种实用的IT技术文章
    - ✓ iOS、前端、后台、人工智能、区块链、逆向破解、数据结构与算法等
  - 招聘、内推信息

更多学习资料

- iOS底层原理班（上）
  - 逆向工程
  - https://ke.qq.com/course/314070

- iOS底层原理班（下）
  - 面试题
  - https://ke.qq.com/course/314526

- 目前已开设3期
  - 学习人数1500人左右
  - 最近才开始放到腾讯课堂

- 浅谈高级IT技术点的学习要素
  - 英语好是很大的优势
  - 知识储备要足够
  - 学习兴趣+上进心

- 个人计划
  - 课程、文章、书
  - 涵盖编程语言
    - ✓ OC、Swift、JS、Python、C++、Java、Go等