Clang 内部实现

邢明杰
mingjie.xing@gmail.com

关于本幻灯片

- 源代码基于 llvm/clang 3.5.0

- UML 图使用 plantUML + LibreOffice Writer 制作

- 参考了 doxygen 生成的类继承关系图

- 使用 Screenshots 截图

目录

- 总体介绍
- 驱动器
- 前端
- 核心库
- 应用库 (略)

Clang 是什么

- LLVM 的 C, C++, Objective C, Objective C++ 前端

- LLVM 的本地（LLVM native）编译器
  - 扮演和 gcc 类似的 driver 功能
  - 集成了类似 cc1 （gcc 的编译器）的编译器功能
  - 集成了类似 as （gnu 的汇编器）的汇编器功能

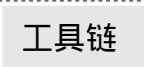- 还可以作为库，被其它工具（例如 clang static analyzer）使用

代码布局

- 核心代码作为库

- 模块化

```
clang
├── bindings
├── CMakeLists.txt
├── CODE_OWNERS.TXT
├── docs
├── examples
├── include
├── INPUTS
├── INSTALL.txt
├── lib
├── LICENSE.TXT
├── Makefile
├── ModuleInfo.txt
├── NOTES.txt
├── README.txt
├── runtime
├── test
├── tools
├── unittests
├── utils
└── www
```

```
clang/lib/
├── Analysis
├── ARCMigrate
├── AST
├── ASTMatchers
├── Basic
├── CMakeLists.txt
├── CodeGen
├── Driver
├── Edit
├── Format
├── Frontend
├── FrontendTool
├── Headers
├── Index
├── Lex
├── Makefile
├── Parse
├── Rewrite
├── Sema
├── Serialization
├── StaticAnalyzer
└── Tooling
```

```
clang/tools/
├── arcmt-test
├── c-arcmt-test
├── c-index-test
├── clang-check
├── clang-format
├── clang-format-vs
├── CMakeLists.txt
├── diag-build
├── diagtool
├── driver
├── libclang
├── Makefile
├── scan-build
└── scan-view
```

驱动器

驱动器

工具链

驱动器

源文件 → 编译器 → 汇编文件 → 汇编器 → 目标文件 → 连接器 → 可执行程序

-v 选项

$ clang -o foo foo.c -v

"/usr/bin/clang -cc1 -triple x86_64-pc-linux-gnu -emit-obj -mrelax-all -disable-free -disable-llvm-verifier -main-file-name foo.c -mrelocation-model static -mdisable-fp-elim -fmath-errno -masm-verbose -mconstructor-aliases -munwind-tables -fuse-init-array -target-cpu x86-64 -target-linker-version 2.24 -v -resource-dir /usr/bin/../lib/clang/3.4 -cxx-isystem . -internal-isystem /usr/local/include -internal-isystem /usr/bin/../lib/clang/3.4/include -internal-externc-isystem /usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/include -internal-externc-isystem /usr/include/x86_64-linux-gnu -internal-externc-isystem /include -internal-externc-isystem /usr/include -ferror-limit 19 -fmessage-length 80 -mstackrealign -fobjc-runtime=gcc -fdiagnostics-show-option -fcolor-diagnostics -vectorize-slp -o /tmp/foo-abd1a4.o -x c foo.c

"/usr/bin/ld" -z relro --hash-style=gnu --build-id --eh-frame-hdr -m elf_x86_64 -dynamic-linker /lib64/ld-linux-x86-64.so.2 -o foo /usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/crt1.o /usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/crti.o /usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/crtbegin.o -L/usr/bin/../lib/gcc/x86_64-linux-gnu/4.8 -L/usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu -L/lib/x86_64-linux-gnu -L/lib/../lib64 -L/usr/lib/x86_64-linux-gnu -L/usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/../../.. -L/lib -L/usr/lib /tmp/foo-abd1a4.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed -lgcc_s --no-as-needed /usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/crtend.o /usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/crtn.o

main 函数

第一个参数为" -cc1" ?

第一个参数为" -cc1as" ?
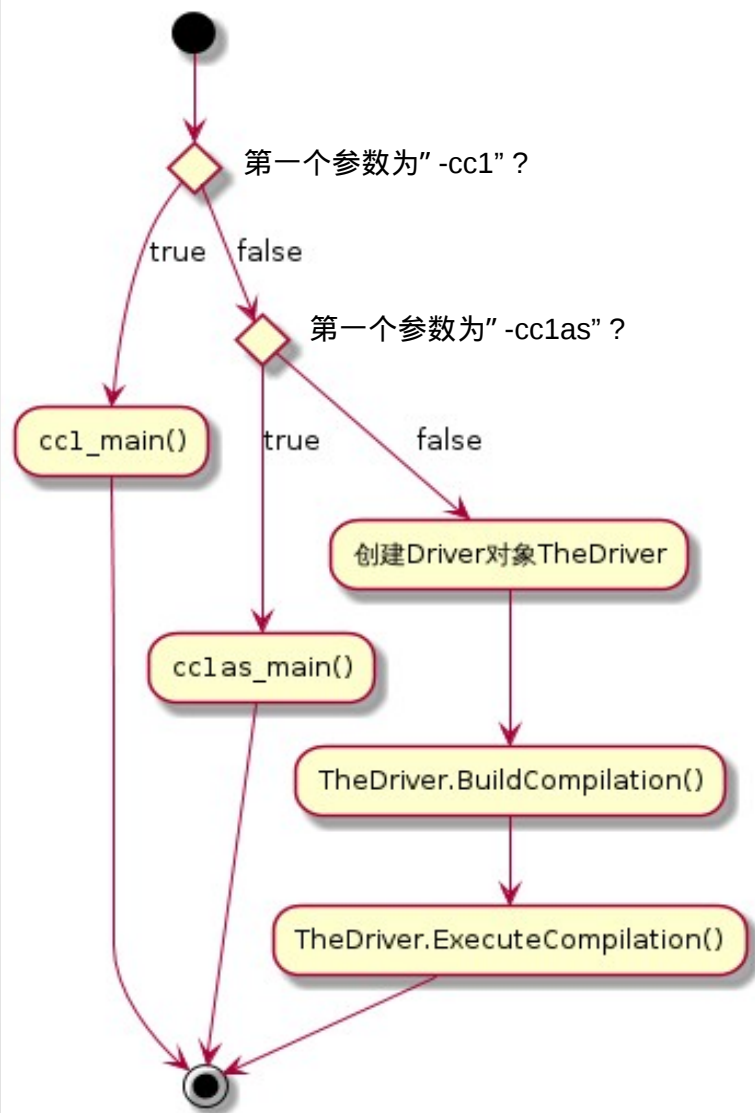
```cpp
// Handle -cc1 integrated tools.
if (argv.size() > 1 && StringRef(argv[1]).startswith("-cc1")) {
  StringRef Tool = argv[1] + 4;

  if (Tool == "")
    return cc1_main(argv.data()+2, argv.data()+argv.size(), argv[0],
                    (void*) (intptr_t) GetExecutablePath);
  if (Tool == "as")
    return cc1as_main(argv.data()+2, argv.data()+argv.size(), argv[0],
                      (void*) (intptr_t) GetExecutablePath);

  // Reject unknown tools.
  llvm::errs() << "error: unknown integrated tool '" << Tool << "'\n";
  return 1;
}
```
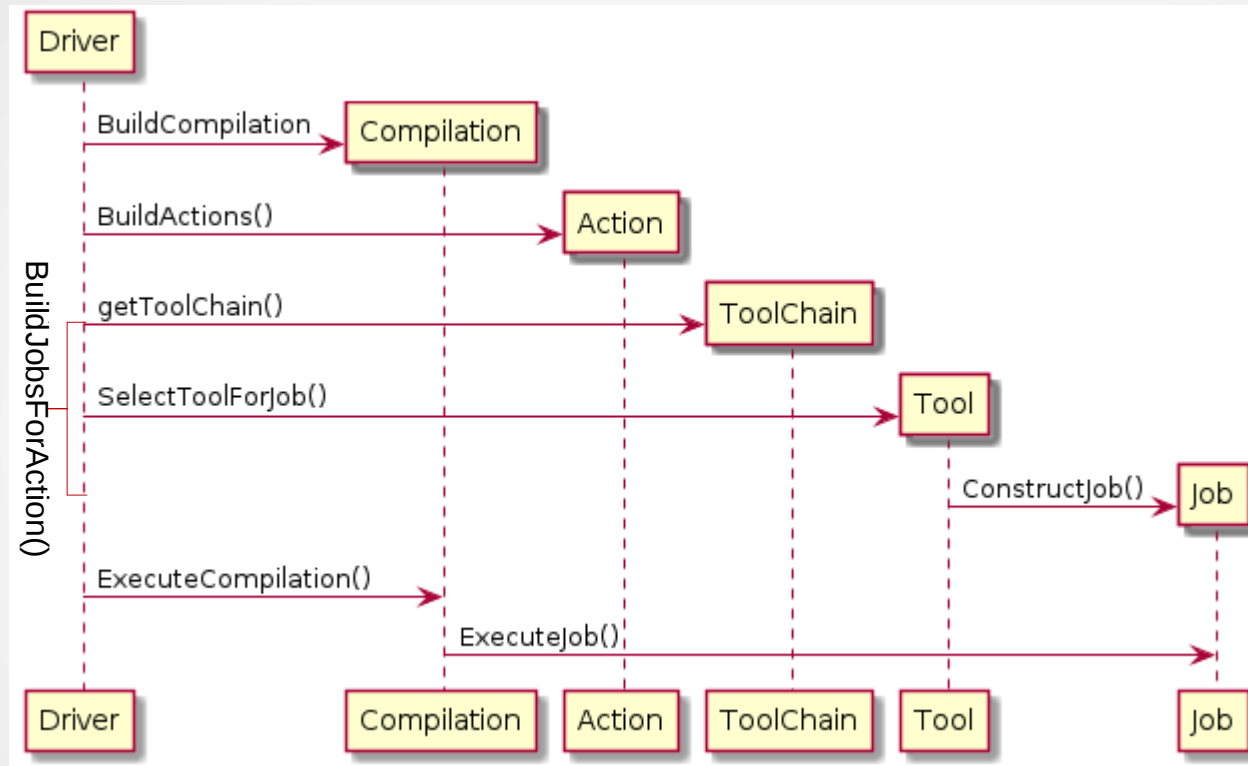
tools/driver/driver.cpp

主要的 class

- Driver

- Compilation
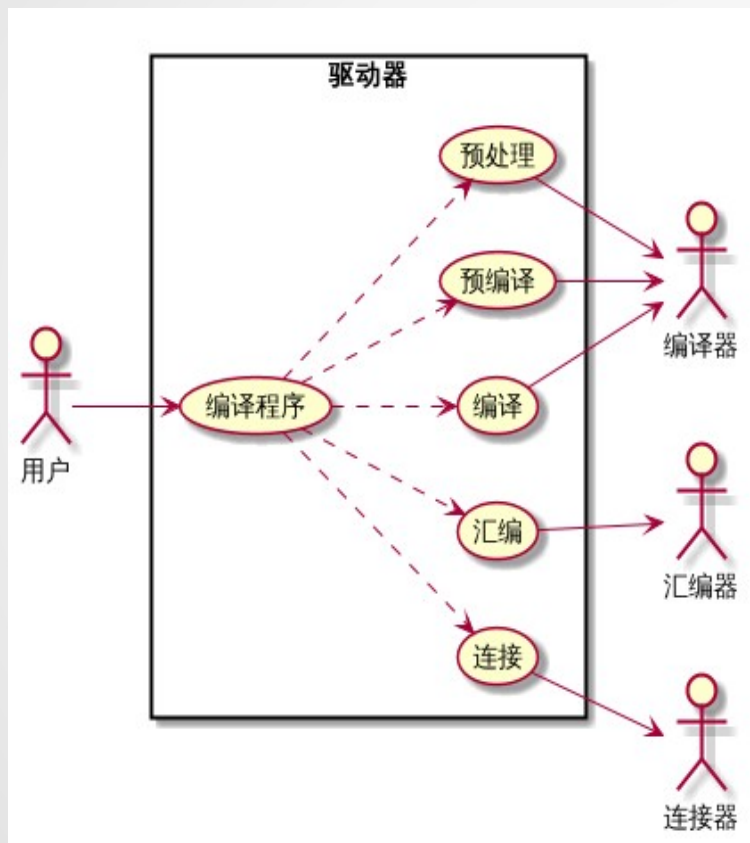
- Action

- Job

- ToolChain, Tool

Driver, Compilation, ...

Driver - Encapsulate logic for constructing compilation processes from a set of gcc-driver-like command line arguments.

Compilation - A set of tasks to perform for a single driver invocation.

Phases


```cpp
namespace phases {
  /// ID - Ordered values for successive stages in the
  /// compilation process which interact with user options.
  enum ID {
    Preprocess,
    Precompile,
    Compile,
    Assemble,
    Link
  };

  enum {
    MaxNumberOfPhases = Link + 1
  };

  const char *getPhaseName(ID Id);

} // end namespace phases
```

include/clang/Driver/Phases.h

Action, JobAction
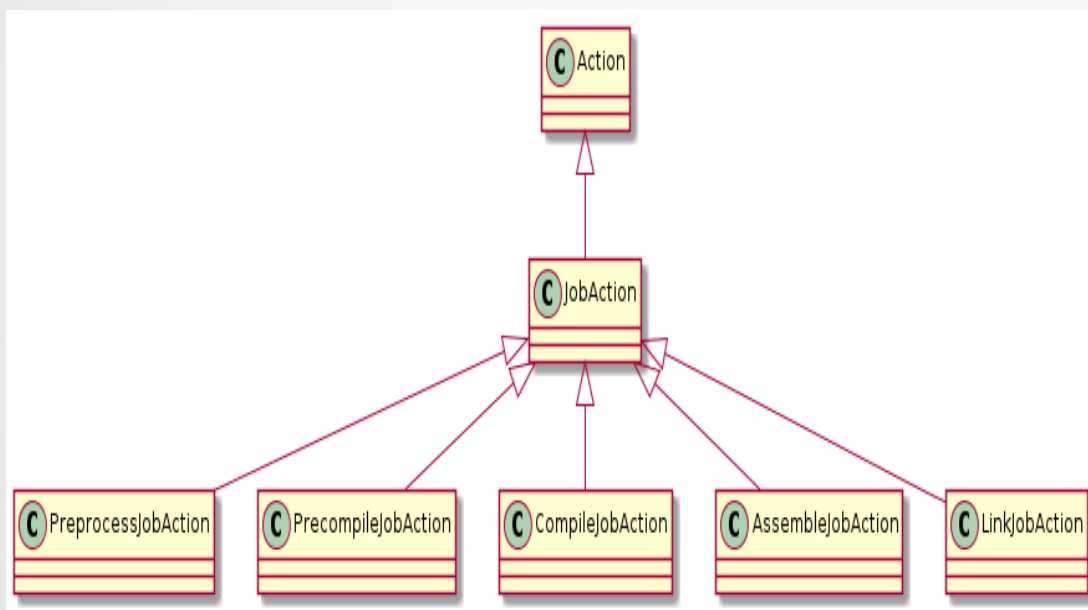

```
enum ActionClass {
  InputClass = 0,
  BindArchClass,
  PreprocessJobClass,
  PrecompileJobClass,
  AnalyzeJobClass,
  MigrateJobClass,
  CompileJobClass,  ▷
  AssembleJobClass, ▷
  LinkJobClass, ▷
  LipoJobClass,
  DsymutilJobClass,
  VerifyDebugInfoJobClass,
  VerifyPCHJobClass,

  JobClassFirst=PreprocessJobClass,
  JobClassLast=VerifyPCHJobClass
};
```

include/clang/Driver/Action.h

Action - Represent an abstract compilation step to perform.

Job, Command
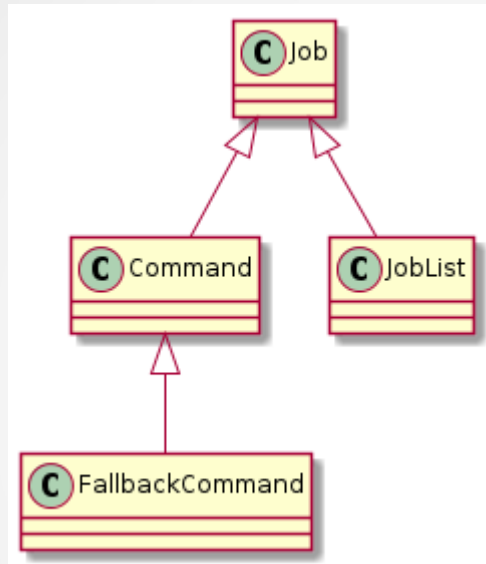

```
enum JobClass {
    CommandClass,
    FallbackCommandClass,
    JobListClass
};
```
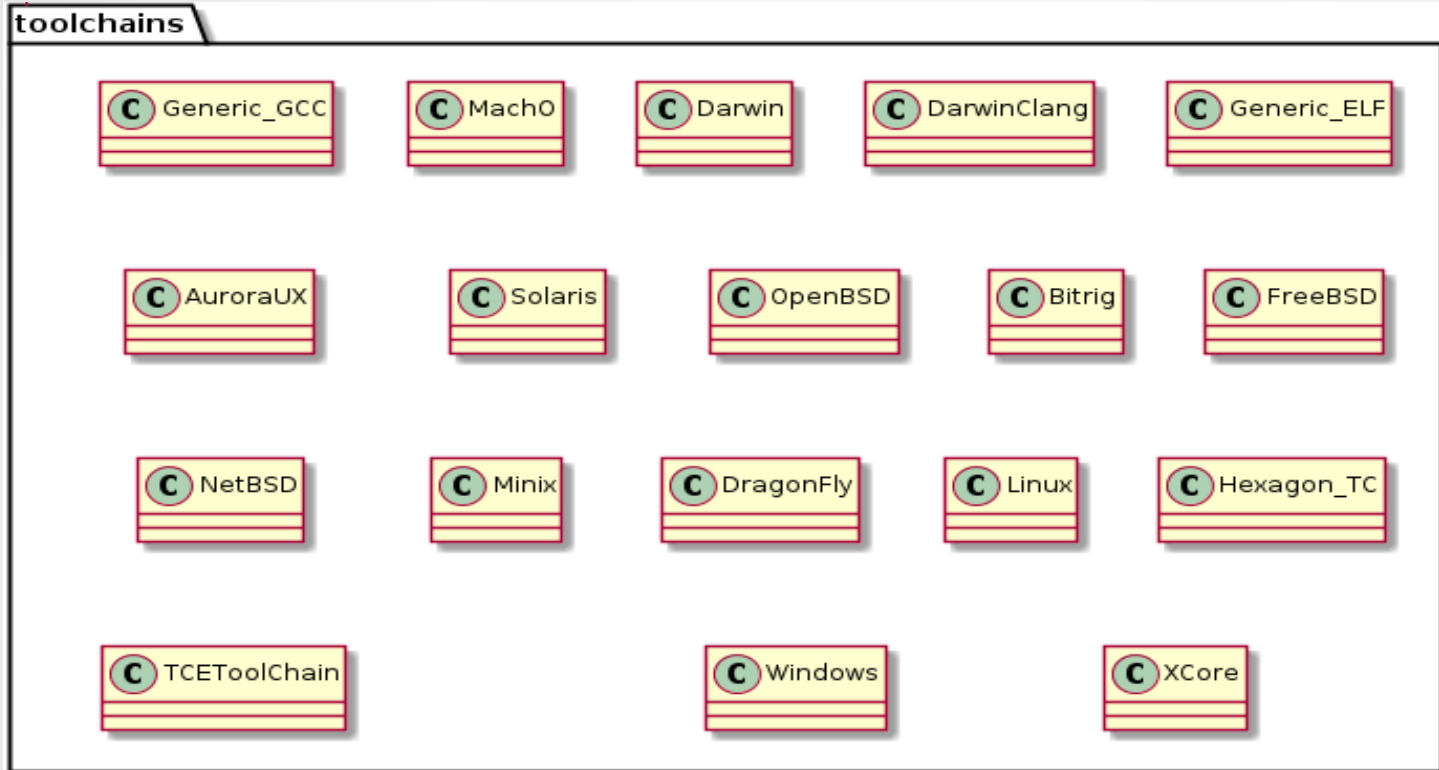
include/clang/Driver/Job.h

Command - An executable path/name and argument vector to execute.

FallbackCommand - Like Command, but with a fallback which is executed in case the primary command crashes.

JobList - A sequence of jobs to perform.

ToolChain, Tool

ToolChain - Access to tools for a single platform.

Tool - Information on a specific compilation tool.

命令行选项

文件： include/clang/Driver/Options.td
This file defines the options accepted by clang.

文件： include/clang/Driver/CC1Options.td
This file defines the options accepted by clang -cc1 and clang -cc1as.

```cpp
// Invoke ourselves in -cc1 mode.
//
// FIXME: Implement custom jobs for internal actions.
CmdArgs.push_back("-cc1");

// Add the "effective" target triple.
CmdArgs.push_back("-triple");
std::string TripleStr = getToolChain().ComputeEffectiveClangTriple(Args);
CmdArgs.push_back(Args.MakeArgString(TripleStr));
```
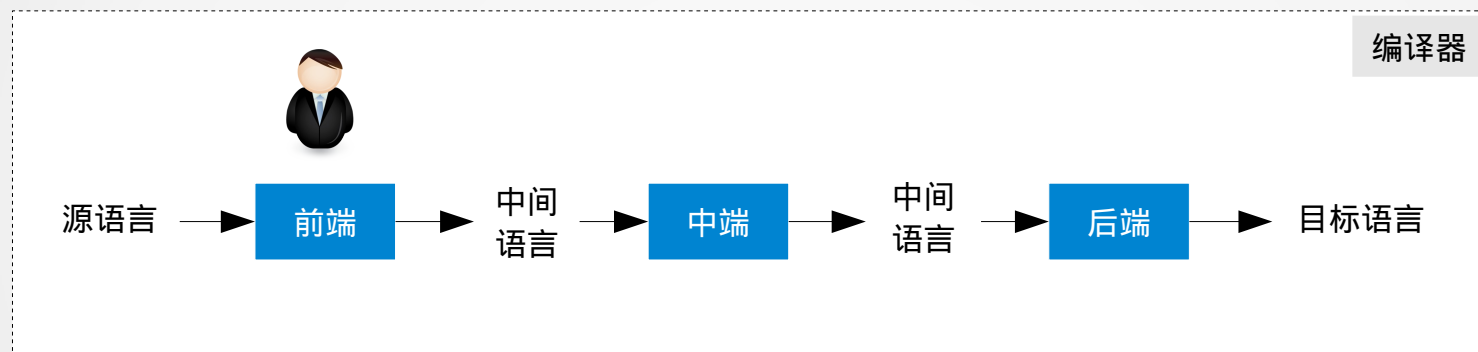
lib/Driver/Tools.cpp

在函数 Clang::ConstructJob() 中，构建 Frontend 的命令行参数。

前端

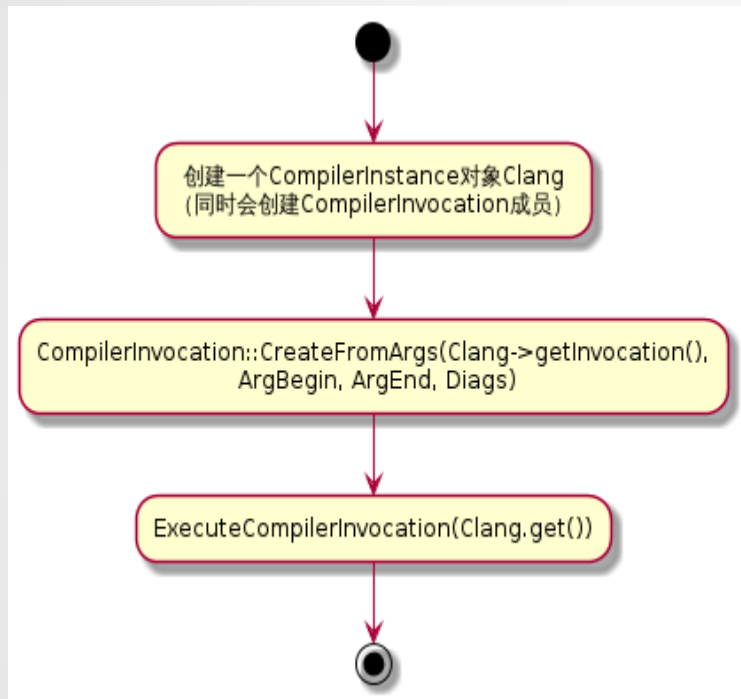# 前端

源语言 → 前端 → 中间语言 → 中端 → 中间语言 → 后端 → 目标语言

编译器

-v 选项

```
$ clang -o foo foo.c -v

"/usr/bin/clang" -cc1 -triple x86_64-pc-linux-gnu -emit-obj -mrelax-all -disable-free
-disable-llvm-verifier -main-file-name foo.c -mrelocation-model static -mdisable-fp-elim
-fmath-errno -masm-verbose -mconstructor-aliases -munwind-tables -fuse-init-array
-target-cpu x86-64 -target-linker-version 2.24 -v -resource-dir /usr/bin/../lib/clang/3.4
-cxx-isystem . -internal-isystem /usr/local/include -internal-isystem
/usr/bin/../lib/clang/3.4/include -internal-externc-isystem /usr/bin/../lib/gcc/x86_64-linux-
gnu/4.8/include -internal-externc-isystem /usr/include/x86_64-linux-gnu -internal-
externc-isystem /include -internal-externc-isystem /usr/include -ferror-limit 19
-fmessage-length 80 -mstackrealign -fobjc-runtime=gcc -fdiagnostics-show-option
-fcolor-diagnostics -vectorize-slp -o /tmp/foo-abd1a4.o -x c foo.c

"/usr/bin/ld" -z relro --hash-style=gnu --build-id --eh-frame-hdr -m elf_x86_64
-dynamic-linker /lib64/ld-linux-x86-64.so.2 -o foo /usr/bin/../lib/gcc/x86_64-linux-
gnu/4.8/../../../x86_64-linux-gnu/crt1.o /usr/bin/../lib/gcc/x86_64-linux-
gnu/4.8/../../../x86_64-linux-gnu/crti.o /usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/crtbegin.o
-L/usr/bin/../lib/gcc/x86_64-linux-gnu/4.8 -L/usr/bin/../lib/gcc/x86_64-linux-
gnu/4.8/../../../x86_64-linux-gnu -L/lib/x86_64-linux-gnu -L/lib/../lib64 -L/usr/lib/x86_64-
linux-gnu -L/usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/../../.. -L/lib -L/usr/lib /tmp/foo-
abd1a4.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as-needed -lgcc_s --no-
as-needed /usr/bin/../lib/gcc/x86_64-linux-gnu/4.8/crtend.o /usr/bin/../lib/gcc/x86_64-
linux-gnu/4.8/../../../x86_64-linux-gnu/crtn.o
```

cc1_main 函数

创建一个CompilerInstance对象Clang
(同时会创建CompilerInvocation成员)

CompilerInvocation::CreateFromArgs(Clang->getInvocation(),
ArgBegin, ArgEnd, Diags)

ExecuteCompilerInvocation(Clang.get())

```
int cc1_main(const char **ArgBegin, const char **ArgEnd,
             const char *Argv0, void *MainAddr) {
  std::unique_ptr<CompilerInstance> Clang(new CompilerInstance());
  IntrusiveRefCntPtr<DiagnosticIDs> DiagID(new DiagnosticIDs());

  // Initialize targets first, so that --version shows registered targets.
  llvm::InitializeAllTargets();
  llvm::InitializeAllTargetMCs();
  llvm::InitializeAllAsmPrinters();
  llvm::InitializeAllAsmParsers();
```

tools/driver/cc1_main.cpp
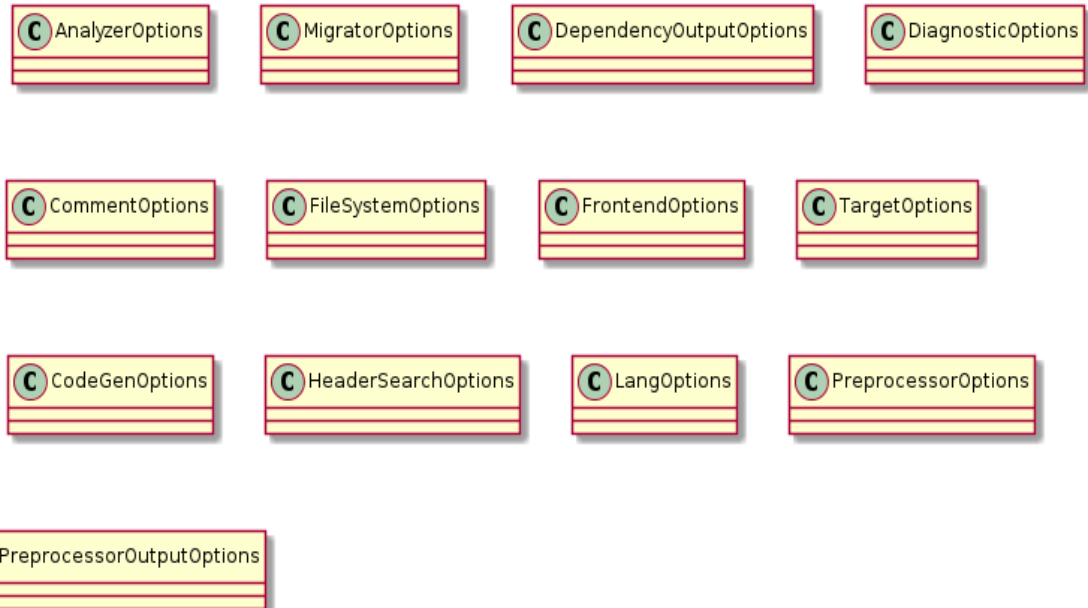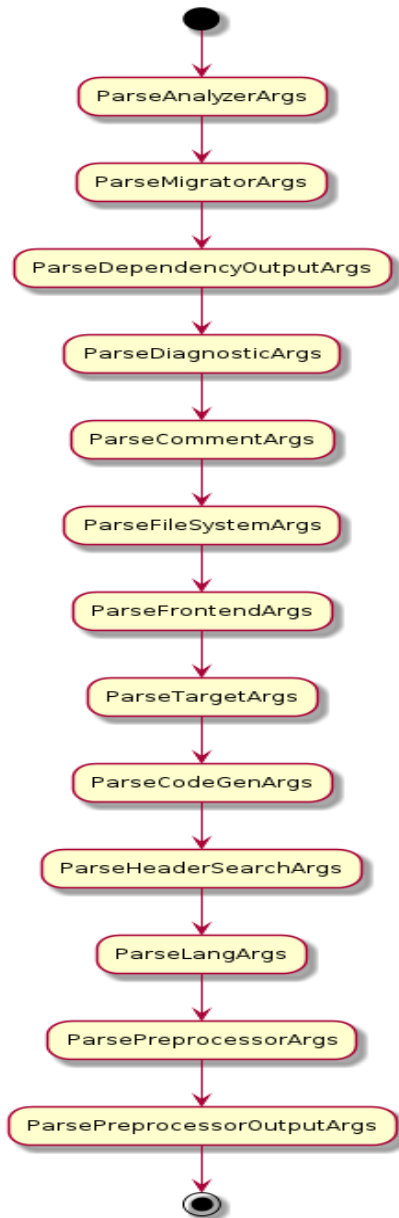
CompilerInstance - Helper class for managing a single instance of the Clang compiler.

CompilerInvocation - Helper class for holding the data necessary to invoke the compiler.

主要的 class

- CompilerInstance

- CompilerInvocation

- FrontendOptions, LangOptions, ...

- FrontendAction

CompilerInvocation::CreateFromArgs

$ find include/ -name *Options.h

lib/Frontend/CompilerInvocation.cpp

clang::ExecuteCompilerInvocation
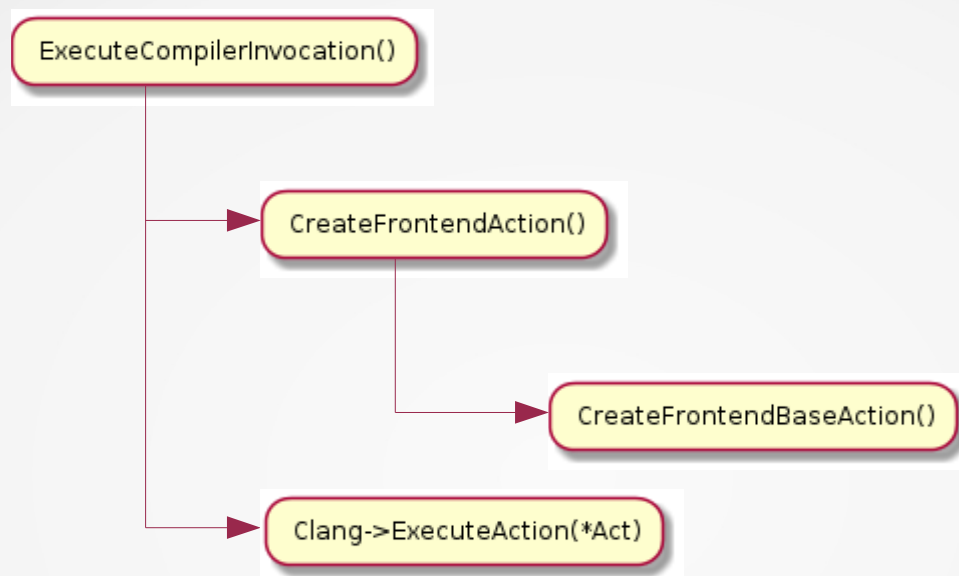
bool clang::ExecuteCompilerInvocation(CompilerInstance *Clang) {

```cpp
// Create and execute the frontend action.
std::unique_ptr<FrontendAction> Act(CreateFrontendAction(*Clang));
if (!Act)
  return false;
bool Success = Clang->ExecuteAction(*Act);
if (Clang->getFrontendOpts().DisableFree)
  BuryPointer(Act.release());
return Success;
```

lib/FrontendTool/ExecuteCompilerInvocation.cpp

CreateFrontendBaseAction

```
switch (CI.getFrontendOpts().ProgramAction) {
case ASTDeclList:          return new ASTDeclListAction();
case ASTDump:              return new ASTDumpAction(); ⊿
case ASTPrint:             return new ASTPrintAction();
case ASTView:              return new ASTViewAction();
case DumpRawTokens:        return new DumpRawTokensAction();
case DumpTokens:           return new DumpTokensAction();
case EmitAssembly:         return new EmitAssemblyAction(); ⊿
case EmitBC:               return new EmitBCAction(); ⊿
case EmitHTML:             return new HTMLPrintAction();
case EmitLLVM:             return new EmitLLVMAction();
case EmitLLVMOnly:         return new EmitLLVMOnlyAction();
case EmitCodeGenOnly:      return new EmitCodeGenOnlyAction();
case EmitObj:              return new EmitObjAction();
case FixIt:                return new FixItAction();
case GenerateModule:       return new GenerateModuleAction;
case GeneratePCH:          return new GeneratePCHAction;
case GeneratePTH:          return new GeneratePTHAction();
case InitOnly:             return new InitOnlyAction();
case ParseSyntaxOnly:      return new SyntaxOnlyAction();
case ModuleFileInfo:       return new DumpModuleInfoAction();
case VerifyPCH:            return new VerifyPCHAction();
```
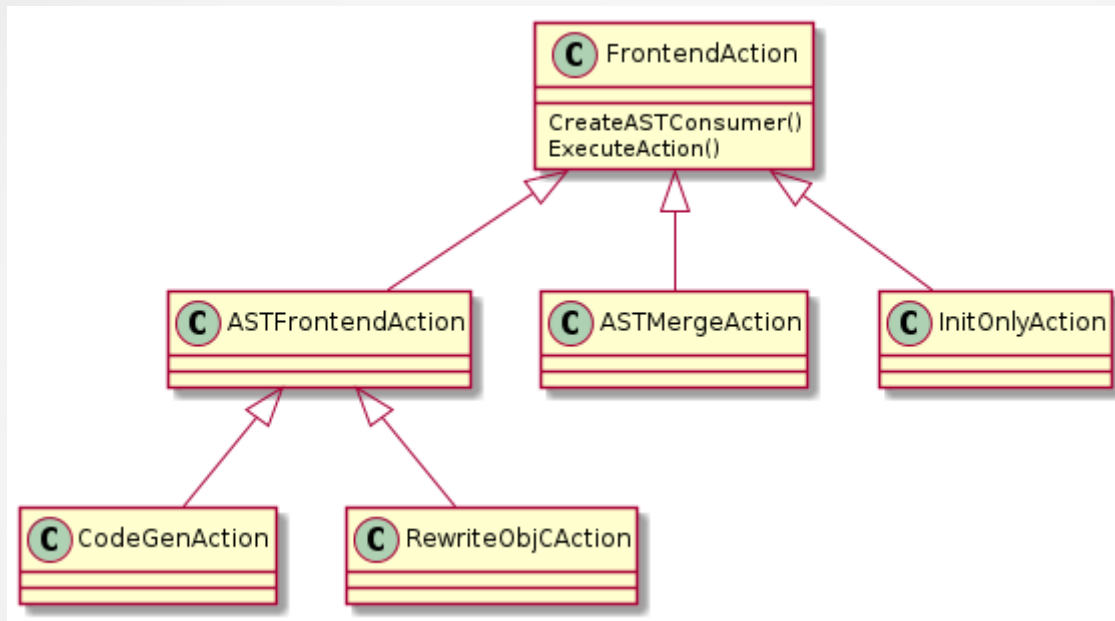
lib/FrontendTool/ExecuteCompilerInvocation.cpp

FrontendAction

FrontendAction - Abstract base class for actions which can be performed by the frontend.

ASTFrontendAction - Abstract base class to use for AST consumer-based frontend actions.

ASTFrontendAction::ExecuteAction()

```cpp
void ASTFrontendAction::ExecuteAction() {
  CompilerInstance &CI = getCompilerInstance();
  if (!CI.hasPreprocessor())
    return;

  // FIXME: Move the truncation aspect of this into Sema, we delayed this till
  // here so the source manager would be initialized.
  if (hasCodeCompletionSupport() &&
      !CI.getFrontendOpts().CodeCompletionAt.FileName.empty())
    CI.createCodeCompletionConsumer();

  // Use a code completion consumer?
  CodeCompleteConsumer *CompletionConsumer = nullptr;
  if (CI.hasCodeCompletionConsumer())
    CompletionConsumer = &CI.getCodeCompletionConsumer();

  if (!CI.hasSema())
    CI.createSema(getTranslationUnitKind(), CompletionConsumer);

  ParseAST(CI.getSema(), CI.getFrontendOpts().ShowStats,
           CI.getFrontendOpts().SkipFunctionBodies);
}
```
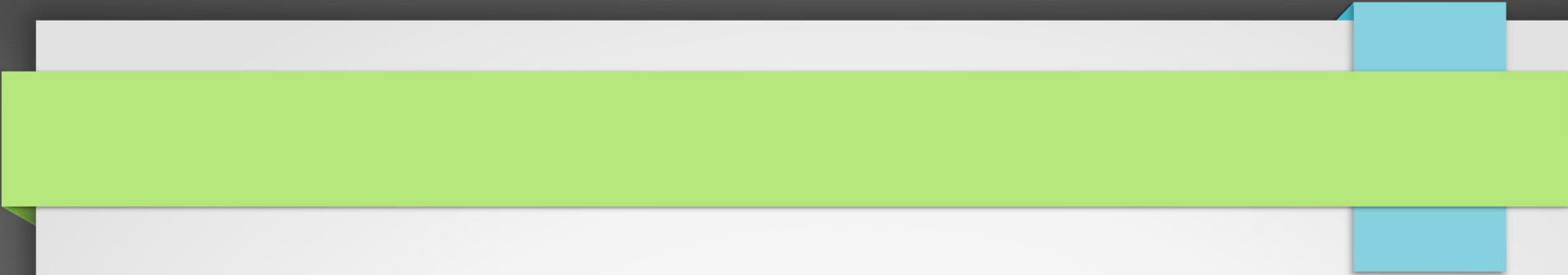
lib/Frontend/FrontendAction.cpp

ASTFrontendAction::ExecuteAction()

ParseAST(...)

核心库

ParseAST
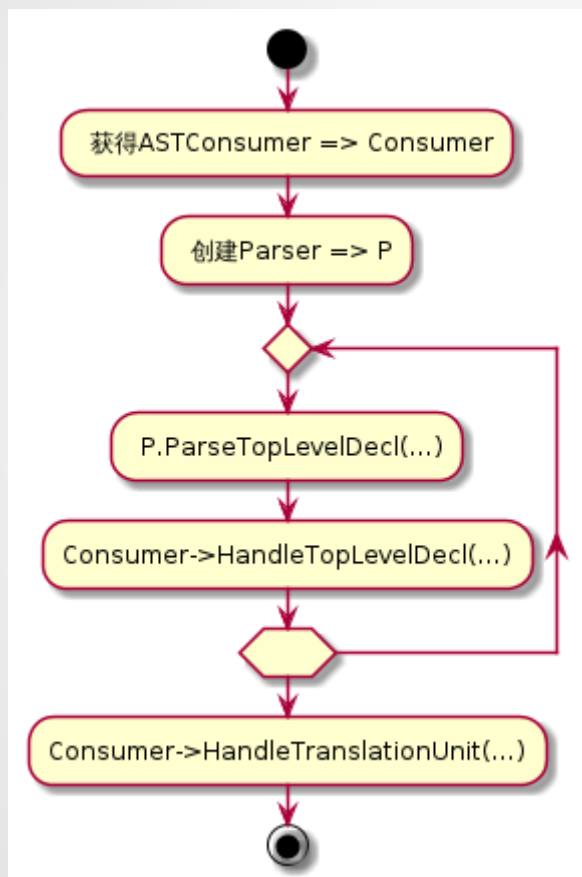

```
if (P.ParseTopLevelDecl(ADecl)) {
  if (!External && !S.getLangOpts().CPlusPlus)
    P.Diag(diag::ext_empty_translation_unit);
} else {
  do {
    // If we got a null return and something *was* parsed, ignore it.  This
    // is due to a top-level semicolon, an action override, or a parse error
    // skipping something.
    if (ADecl && !Consumer->HandleTopLevelDecl(ADecl.get()))
      return;
  } while (!P.ParseTopLevelDecl(ADecl));
}

// Process any TopLevelDecls generated by #pragma weak.
for (SmallVectorImpl<Decl *>::iterator
     I = S.WeakTopLevelDecls().begin(),
     E = S.WeakTopLevelDecls().end(); I != E; ++I)
  Consumer->HandleTopLevelDecl(DeclGroupRef(*I));

Consumer->HandleTranslationUnit(S.getASTContext());
```

lib/Parse/ParseAST.cpp

词法、语法、语义、 AST
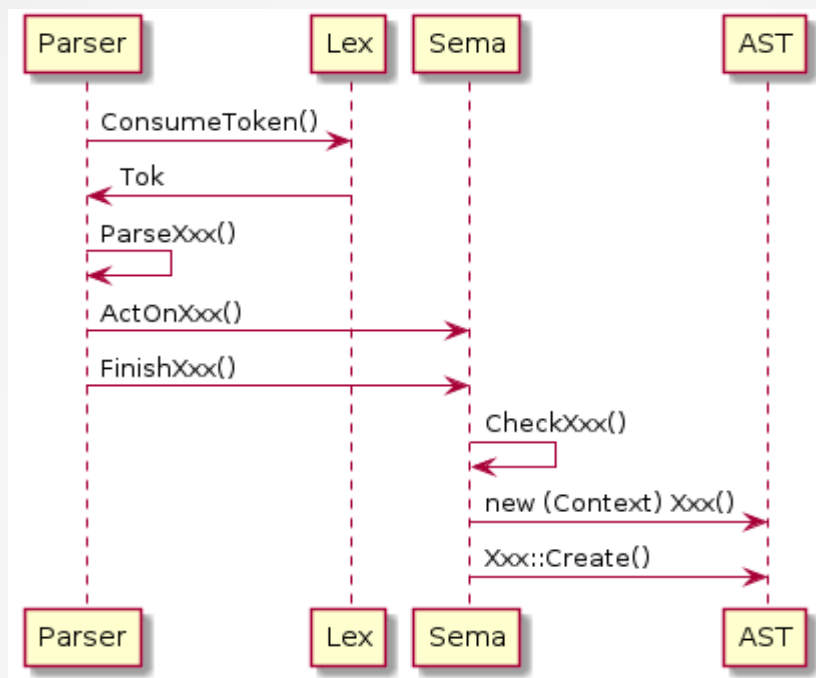
Lexer - This provides a simple interface that turns a text buffer into a stream of tokens.

Parser - This implements a parser for the C family of languages.

Sema - This implements semantic analysis and AST building for C.

AST 包括： Type 、 Decl 、 Stmt 、 Expr

Token

预处理关键字

pragma 注解

```
PPKEYWORD(if)
PPKEYWORD(ifdef)
PPKEYWORD(ifndef)
PPKEYWORD(elif)
PPKEYWORD(else)
PPKEYWORD(endif)
PPKEYWORD(defined)
```

```
// Annotations for OpenMP pragma directives - #pragma omp ...
// The lexer produces these so that they only take effect when the parser
// handles #pragma omp ... directives.
ANNOTATION(pragma_openmp)
ANNOTATION(pragma_openmp_end)
```

```
KEYWORD(auto                           , KEYALL)
KEYWORD(break                          , KEYALL)
KEYWORD(case                           , KEYALL)
KEYWORD(char                           , KEYALL)
KEYWORD(const                          , KEYALL)
KEYWORD(continue                       , KEYALL)
KEYWORD(default                        , KEYALL)
KEYWORD(do                             , KEYALL)
```

```
PUNCTUATOR(l_square,          "[")
PUNCTUATOR(r_square,          "]")
PUNCTUATOR(l_paren,           "(")
PUNCTUATOR(r_paren,           ")")
PUNCTUATOR(l_brace,           "{")
PUNCTUATOR(r_brace,           "}")
PUNCTUATOR(period,            ".")
PUNCTUATOR(ellipsis,          "...")
PUNCTUATOR(amp,               "&")
PUNCTUATOR(ampamp,            "&&")
PUNCTUATOR(ampequal,          "&=")
```

语言关键字

标点符号

include/clang/Basic/TokenKinds.def

语法分析

```
/// ParseExternalDeclaration:
///
///       external-declaration: [C99 6.9], declaration: [C++ dcl.dcl]
///         function-definition
///         declaration
/// [GNU]   asm-definition
/// [GNU]   __extension__ external-declaration
/// [OBJC]  objc-class-definition
/// [OBJC]  objc-class-declaration
/// [OBJC]  objc-alias-declaration
/// [OBJC]  objc-protocol-definition
/// [OBJC]  objc-method-definition
/// [OBJC]  @end
/// [C++]   linkage-specification
/// [GNU] asm-definition:
///         simple-asm-expr ';'
/// [C++11] empty-declaration
/// [C++11] attribute-declaration
///
/// [C++11] empty-declaration:
///           ';'
///
/// [C++0x/GNU] 'extern' 'template' declaration
Parser::DeclGroupPtrTy
Parser::ParseExternalDeclaration(ParsedAttributesWithRange &attrs,
                        ParsingDeclSpec *DS) {
```

lib/Parse/Parser.cpp

ISO/IEC 9899:TC3   Committee Draft — Septermber 7, 2007

6.9 External definitions
Syntax

    translation-unit:
        external-declaration
        translation-unit external-declaration

    external-declaration:
        function-definition
        declaration

C 语言标准规范

语言标准规范 + 递归下降分析

语法分析

```
Decl *Parser::ParseFunctionStatementBody(Decl *Decl, ParseScope &BodyScope) {
  assert(Tok.is(tok::l_brace));
  SourceLocation LBraceLoc = Tok.getLocation();

  if (SkipFunctionBodies && (!Decl || Actions.canSkipFunctionBody(Decl)) &&
      trySkippingFunctionBody()) {
    BodyScope.Exit();
    return Actions.ActOnSkippedFunctionBody(Decl);
  }
}
```

lib/Parse/ParseStmt.cpp

$ gdb -args "/home/xmj/install/cap-llvm-3.5/bin/clang" -cc1 …
(gdb) b clang::Parser::ParseFunctionStatementBody
(gdb) r

```
(gdb) p Tok
$1 = {
  Loc = {
    ID = 24
  },
  UintData = 1,
  PtrData = 0x0,
  Kind = clang::tok::l_brace,
  Flags = 1 '\001'
}
```

语义分析

```
static const int volatile x, *y, *(*(*z)[10])(const void *x);
-----------------------  -  --  ---------------------------
  declaration-specifiers  \  |   /
                            declarators
```

class DeclSpec
class Declarator
include/clang/Sema/DeclSpec.h

```
/// \brief Convert the specified declspec to the appropriate type
/// object.
/// \param state Specifies the declarator containing the declaration specifier
/// to be converted, along with other associated processing state.
/// \returns The type described by the declaration specifiers.  This function
/// never returns null.
static QualType ConvertDeclSpecToType(TypeProcessingState &state) {
  // FIXME: Should move the logic from DeclSpec::Finish to here for validity
  // checking.

  Sema &S = state.getSema();
  Declarator &declarator = state.getDeclarator();
  const DeclSpec &DS = declarator.getDeclSpec();
```

lib/Sema/SemaType.cpp

语义分析

```
StmtResult
Sema::ActOnLabelStmt(SourceLocation IdentLoc, LabelDecl *TheDecl,
                     SourceLocation ColonLoc, Stmt *SubStmt) {
  // If the label was multiply defined, reject it now.
  if (TheDecl->getStmt()) {
    Diag(IdentLoc, diag::err_redefinition_of_label) << TheDecl->getDeclName();
    Diag(TheDecl->getLocation(), diag::note_previous_definition);
    return SubStmt;
  }
}
```

lib/Sema/SemaStmt.cpp

```
def warn_missing_braces : Warning<
  "suggest braces around initialization of subobject">,
  InGroup<MissingBraces>, DefaultIgnore;

def err_redefinition_of_label : Error<"redefinition of label %0">;
```

include/clang/Basic/DiagnosticSemaKinds.td

AST

$ clang -Xclang -ast-dump test.c

```c
int add(int a, int b)
{
    int c;
    c = a + b;
    return c;
}
```

```
TranslationUnitDecl 0x6cdced0 <<invalid sloc>> <invalid sloc>
|-TypedefDecl 0x6cdd3d0 <<invalid sloc>> <invalid sloc> implicit __int128_t '__int128'
|-TypedefDecl 0x6cdd430 <<invalid sloc>> <invalid sloc> implicit __uint128_t 'unsigned __int128'
|-TypedefDecl 0x6cdd780 <<invalid sloc>> <invalid sloc> implicit __builtin_va_list '__va_list_tag [1]'
`-FunctionDecl 0x6cdd920 <test.c:1:1, line:6:1> line:1:5 add 'int (int, int)'
  |-ParmVarDecl 0x6cdd7e0 <col:9, col:13> col:13 used a 'int'
  |-ParmVarDecl 0x6cdd850 <col:16, col:20> col:20 used b 'int'
  `-CompoundStmt 0x6d1a788 <line:2:1, line:6:1>
    |-DeclStmt 0x6cdda38 <line:3:3, col:8>
    | `-VarDecl 0x6cdd9e0 <col:3, col:7> col:7 used c 'int'
    |-BinaryOperator 0x6cddb90 <line:4:3, col:11> 'int' '='
    | |-DeclRefExpr 0x6cdda60 <col:3> 'int' lvalue Var 0x6cdd9e0 'c' 'int'
    | `-BinaryOperator 0x6cddb58 <col:7, col:11> 'int' '+'
    |   |-ImplicitCastExpr 0x6cddb08 <col:7> 'int' <LValueToRValue>
    |   | `-DeclRefExpr 0x6cdda98 <col:7> 'int' lvalue ParmVar 0x6cdd7e0 'a' 'int'
    |   `-ImplicitCastExpr 0x6cddb30 <col:11> 'int' <LValueToRValue>
    |     `-DeclRefExpr 0x6cddad0 <col:11> 'int' lvalue ParmVar 0x6cdd850 'b' 'int'
    `-ReturnStmt 0x6d1a758 <line:5:3, col:10>
      `-ImplicitCastExpr 0x6d1a730 <col:10> 'int' <LValueToRValue>
        `-DeclRefExpr 0x6cddbc8 <col:10> 'int' lvalue Var 0x6cdd9e0 'c' 'int'
```

Type 和 QualType

```cpp
/// QualType - For efficiency, we don't store CV-qualified types as nodes on
/// their own: instead each reference to a type stores the qualifiers.  This
/// greatly reduces the number of nodes we need to allocate for types (for
/// example we only need one for 'int', 'const int', 'volatile int',
/// 'const volatile int', etc).
///
/// As an added efficiency bonus, instead of making this a pair, we
/// just store the two bits we care about in the low bits of the
/// pointer.  To handle the packing/unpacking, we make QualType be a
/// simple wrapper class that acts like a smart pointer.  A third bit
/// indicates whether there are extended qualifiers present, in which
/// case the pointer points to a special structure.
class QualType {
  // Thankfully, these are efficiently composable.
  llvm::PointerIntPair<llvm::PointerUnion<const Type*,const ExtQuals*>,
                       Qualifiers::FastWidth> Value;
```

include/clang/AST/Type.h

```cpp
class Expr : public Stmt {
  QualType TR;
```

- 在 gdb 中 dump
    - (gdb) p ((DeclRefExpr*)0x59c02f8)->TR.dump()

Decl 和 DeclStmt

```
/// DeclStmt - Adaptor class for mixing declarations with statements and
/// expressions. For example, CompoundStmt mixes statements, expressions
/// and declarations (variables, types). Another example is ForStmt, where
/// the first statement can be an expression or a declaration.
///
class DeclStmt : public Stmt {
  DeclGroupRef DG;
  SourceLocation StartLoc, EndLoc;

public:
  DeclStmt(DeclGroupRef dg, SourceLocation startLoc,
           SourceLocation endLoc) : Stmt(DeclStmtClass), DG(dg),
                                    StartLoc(startLoc), EndLoc(endLoc) {}
```

include/clang/AST/Stmt.h

- 使用 tabelgen 来分别描述 Decl 的类关系

    – include/clang/Basic/DeclNodes.td

Stmt 和 Expr

```
/// Expr - This represents one expression.  Note that Expr's are subclasses of
/// Stmt.  This allows an expression to be transparently used any place a Stmt
/// is required.
///
class Expr : public Stmt {
  QualType TR;

protected:
  Expr(StmtClass SC, QualType T, ExprValueKind VK, ExprObjectKind OK,
       bool TD, bool VD, bool ID, bool ContainsUnexpandedParameterPack)
    : Stmt(SC)
```

include/clang/AST/Expr.h

- 使用 tabelgen 来分别描述 Stmt 、 Expr 的类关系

    – include/clang/Basic/StmtNodes.td

增加一个 Stmt 节点

- 定义、实现class
  - include/clang/AST/Stmt.h 或 StmtXXX.h
  - lib/AST/Stmt.cpp 或 StmtXXX.cpp
- 描述节点继承关系
  - include/clang/Basic/StmtNodes.td
- 其它相关部分：$ grep StmtNodes.inc include/ lib/ -rl
  - include/clang/AST/RecursiveASTVisitor.h
  - include/clang/AST/DataRecursiveASTVisitor.h
  - lib/AST/StmtPrinter.cpp
  - lib/AST/StmtProfile.cpp
  - lib/Sema/TreeTransform.h
  - lib/Serialization/ASTReaderStmt.cpp
  - lib/Serialization/ASTWriterStmt.cpp
  - ...

ASTConsumer
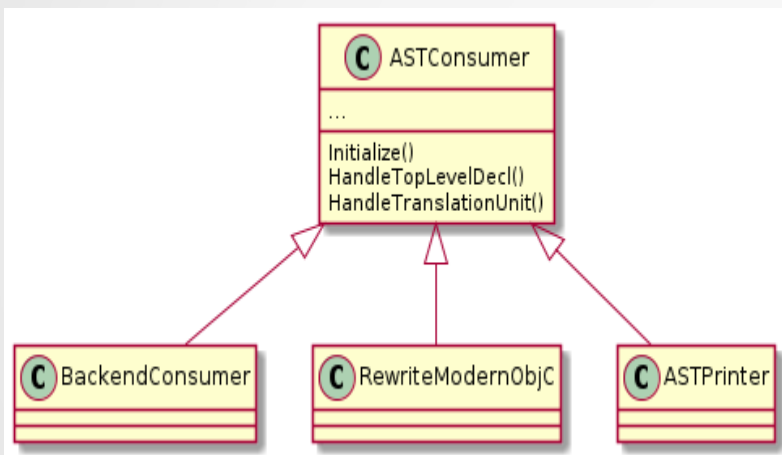

```
/// ASTConsumer - This is an abstract interface that should be implemented by
/// clients that read ASTs.  This abstraction layer allows the client to be
/// independent of the AST producer (e.g. parser vs AST dump file reader, etc).
class ASTConsumer {
  /// \brief Whether this AST consumer also requires information about
  /// semantic analysis.
  bool SemaConsumer;

  friend class SemaConsumer;

public:
  ASTConsumer() : SemaConsumer(false) { }

  virtual ~ASTConsumer() {}

  /// Initialize - This is called to initialize the consumer, providing the
  /// ASTContext.
  virtual void Initialize(ASTContext &Context) {}

  /// HandleTopLevelDecl - Handle the specified top-level declaration.  This is
  /// called by the parser to process every top-level Decl*.
  ///
  /// \returns true to continue parsing, or false to abort parsing.
  virtual bool HandleTopLevelDecl(DeclGroupRef D);
```
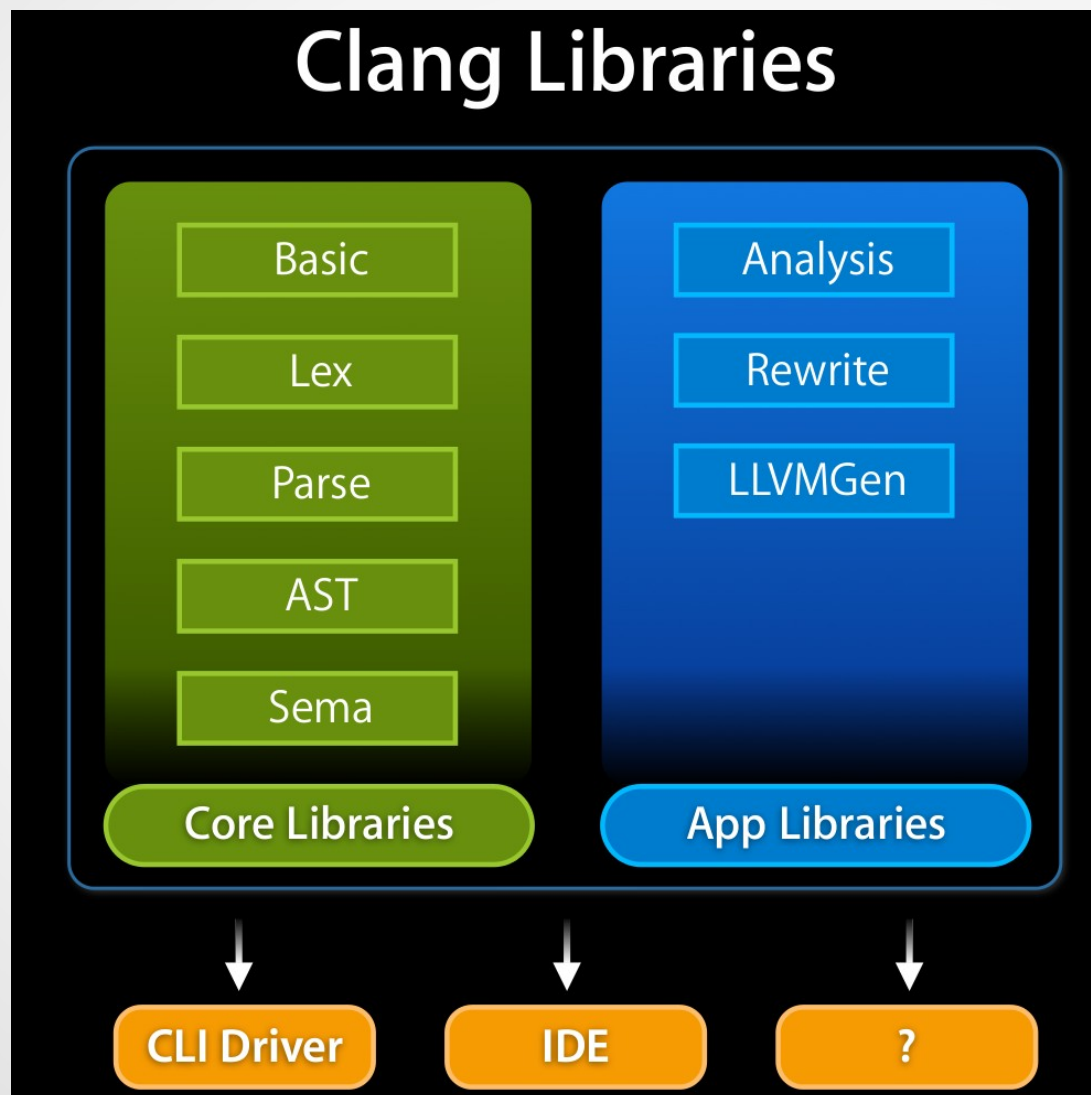
include/clang/AST/ASTConsumer.h

应用库（略）

图片源自

Clang Intro,
Steve Naroff