Boost. 勉強会 #7
clang で入門 解析戦略ー

藤田 典久: @fjnli, id:fjnl

2011/12/03

Agenda

https://github.com/fjnl/boost-study-7-Tokyo/raw/master/sheet.pdf

http://bit.ly/uCnZxE

---

1. About LLVM and clang
2. About libclang
3. 定義元を調べるツールを作ってみる
4. 補足, $+\alpha$

自己紹介

https://github.com/fjnl/boost-study-7-Tokyo/raw/master/sheet.pdf

http://bit.ly/uCnZxE

- 某大学で院生やってます 進路どうしよ
- 研究は HPC とか GPGPU とか
- Twitter → @fjnli
- はてな → id:fjnl
  - C++11 Advent Calender 6 日目
  - Boost Advent Calender 7 日目

LLVM

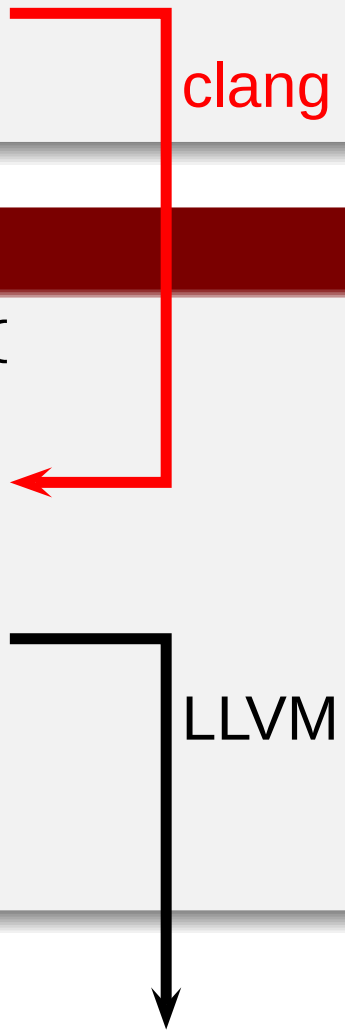
- Low Level Virtual Machine
- `http://llvm.org/`
- 特定の言語，アーキテクチャに依存しない中間言語
- 言語
  - clang, llvm-gcc (C, C++, Obj-C)
  - ghc (Haskell)
  - MacRuby (Ruby) など
- アーキテクチャ
  - x86, x86_64
  - PowerPC
  - ARM など

clang

- http://clang.llvm.org/
- C + Obj-C + C++ コンパイラ期待の新星
- LLVM をバックエンドとしている
- たぶん “クラン” と読む (×クラング)
- BSD License
- Apple (Mac OS/iOS) とか FreeBSD とか

C++

```
int fib(int n) {
  if (n == 0) return 0;
  if (n == 1) return 1;
  return fib(n - 1) + fib(n - 2);
}
```



LLVM Byte Code

```
define i32 @fib(i32 %n) nounwind ssp {
  %1 = alloca i32, align 4
  %2 = alloca i32, align 4
  store i32 %n, i32* %2, align 4
  %3 = load i32* %2, align 4
  %4 = icmp eq i32 %3, 0
  br i1 %4, label %5, label %6
  (略)
}
```

Why?

- gcc のライセンスが GPLv2 → GPLv3 になってしまった
- 最後の GPLv2 の gcc が 4.2.1
- 一部のシステムでは gcc の更新をやめてしまった
  - MacOS
  - *BSD
- 注: バンドルされてないだけで，自分で入れれば最新 gcc は使える

Why?

- 長らく代替コンパイラが不在だった
- Apple → XCode4 からデフォルト
- FreeBSD → 徐々に clang のサポートを拡大中
  - FreeBSD 9 にマージされた
  - 現在 FreeBSD 9.0-RELEASE RC2
  - デフォルトはまだ gcc

さて，ここからは clang のお話…ではありません

さて，ここからは clang のお話…ではありません

libclang がメインです

さて，ここからは clang のお話…ではありません

libclang がメインです

あと Boost は出てきません

libclang

- clang のバックエンドがライブラリ化されたもの
  - 注: 僕が勝手に付けた名前です
- 何ができるの? → C++コードの解析
  - 抽象構文木 (Abstract Syntax Tree; AST) の取得
  - clang static analyzer
  - `http://clang-analyzer.llvm.org/`

libclang

Pros.

- 開発が非常に活発
- コンパイラのコードをそのまま流用できる
- C++11 に対応しつつある

Cons.

- Not user friendly
- 解析のみを欲するにはやや巨大

実験環境

- LLVM 2.9
- clang 2.9
- gcc 4.6.1
- C++ 11

実験環境

- LLVM 2.9
- clang 2.9
- gcc 4.6.1
- C++ 11

- 11 月 30 日に LLVM 3.0 および
- clang 3.0 が出てるはず
- Windows 環境で開発しないのでわかりません．ごめんなさい．

Target

```
$ ./where in.cpp f
  in.cpp:2:5

$ ./where in.cpp g
  in.cpp:6:6
```

in.cpp

```
1: #include <iostream>
2: int f(int x, int y) {
3:   return x + y;
4: }
5:
6: void g() {
7:  std::cout << ``g()'' << std::endl;
8: }
```

コンパイル方法

```
gcc -o where where.cpp \
-D__STDC_CONSTANT_MACROS \
-D__STDC_LIMIT_MACROS \
libLLVM*.a libclang*.a
```

コンパイル方法

```
gcc -o where where.cpp \

-D__STDC_CONSTANT_MACROS \

-D__STDC_LIMIT_MACROS \

libLLVM*.a libclang*.a
```

__STDC_CONSTANT_MACROS

INT8_C, INT16_C, INT32_C, INT64_C, ...

コンパイル方法

```
gcc -o where where.cpp \

-D__STDC_CONSTANT_MACROS \

-D__STDC_LIMIT_MACROS \

libLLVM*.a libclang*.a
```

__STDC_LIMIT_MACROS

INT8_MAX, INT8_MIN,
INT16_MAX, INT16_MIN ...

コンパイル方法

```
gcc -o where where.cpp \

-D__STDC_CONSTANT_MACROS \

-D__STDC_LIMIT_MACROS \

libLLVM*.a libclang*.a
```

lib*.a

libLLVMAnalysis.a,
libLLVMArchive.a
libLLVMAsmParser.a, ...
libclang.a libclangAST.a
libclangAnalysis.a
libclangBasic.a, ...

where.cpp:#include

```
#include <clang/AST/ASTContext.h>
#include <clang/AST/ASTConsumer.h>
#include <clang/AST/Decl.h>
#include <clang/AST/DeclGroup.h>
#include <clang/Basic/TargetInfo.h>
#include <clang/Frontend/CompilerInstance.h>
#include <clang/Frontend/CompilerInvocation.h>
#include <clang/Parse/ParseAST.h>
#include <clang/Lex/Preprocessor.h>
```

where.cpp:main

```
int main(int argc, char** argv) {
  clang::CompilerInstance compiler;
  compiler.createDiagnostics(argc - 1, argv);
  auto& diag = compiler.getDiagnostics();
  auto& invocation = compiler.getInvocation();

  clang::CompilerInvocation::CreateFromArgs(
    invocation, argv + 1, argv + argc - 1, diag);
  compiler.setTarget(clang::TargetInfo::CreateTargetInfo(
    diag, compiler.getTargetOpts()));

  compiler.createFileManager();
  compiler.createSourceManager(compiler.getFileManager());
  compiler.createPreprocessor();
  compiler.createASTContext();
  compiler.setASTConsumer(new consumer(argv[argc - 1]));
  compiler.createSema(false, nullptr);
  (続)
```

where.cpp:main

```
  (続)
  auto& pp = compiler.getPreprocessor();
  pp.getBuiltinInfo().InitializeBuiltins(
      pp.getIdentifierTable(), pp.getLangOptions());

  auto& inputs = compiler.getFrontendOpts().Inputs;
  if (inputs.size() > 0) {
    compiler.InitializeSourceManager(inputs[0].second);
    clang::ParseAST(
      pp,
      &compiler.getASTConsumer(),
      compiler.getASTContext()
    );
  }
  return 0;
}
```

clang::CompilerInstance

```
int main(int argc, char** argv) {
  clang::CompilerInstance compiler;
  compiler.createDiagnostics(argc - 1, argv);
  auto& diag = compiler.getDiagnostics();
  auto& invocation = compi

  clang::CompilerInvocation::CreateFromArgs(
    invocation, argv + 1, argv + argc - 1, diag);
  compiler.setTarget(clang::TargetInfo::CreateTargetInfo(
    diag, compiler.getTargetOpts()));

  compiler.createFileManager();
  compiler.createSourceManager(compiler.getFileManager());
  compiler.createPreprocessor();
  compiler.createASTContext();
  compiler.setASTConsumer(new consumer(argv[argc - 1]));
  compiler.createSema(false, nullptr);
  (続)
```

clang コンパイラ本体みたいなもの

CompilerInvocation::CreateFromArgs

```
int main(int argc, char** argv) {
  clang::CompilerInstance compiler;
  compiler.createDiagnostics(argc - 1, argv);
  auto& diag = compiler.getDiagnostics();
  auto& invocation = compiler.getInvocation();

  clang::CompilerInvocation::CreateFromArgs(
    invocation, argv + 1, argv + argc - 1, diag);
  compiler.setTar                              rgetInfo(
    diag, compil

  compiler.creat
  compiler.creat                               nager());
  compiler.creat
  compiler.creat
  compiler.setAST                              - 1]));
  compiler.createSema(false, nullptr);
  (続)
```

clang 本体の引数解析に
丸投げする.
楽だが不便でもある.

(-D -I -U といったフラグ
を認識させる)

CompilerInvocation::CreateFromArgs

```
clang::CompilerInvocation::CreateFromArgs(
  invocation, argv + 1, argv + argc - 1, diag);
```

※ argc == 1 の時まずいが今回は気にしない方向で.

- 馴染みあるオプションを丸投げできる
  - -I$HOME/boost
  - -DNDEBUG
  - -UFOOBAR ...
- アプリ独自のフラグを渡すのが手間 (see: argv + argc - 1)

CompilerInvocation::CreateFromArgs

```
int main(int argc, char** argv) {
  clang::CompilerInstance compiler;
  compiler.createDiagnostics(argc - 1, argv);
  auto& diag = compiler.getDiagnostics();
  auto& invocation = compiler.getInvocation();

  clang::CompilerInvocation::CreateFromArgs(
    invocation, argv + 1, argv + argc - 1, diag);
  compiler.setTarget(clang::TargetInfo::CreateTargetInfo(
    diag, compiler.getTargetOpts()));

  compiler.createFileManager();
  compiler.createSourceManager(compiler.getFileManager());
  compiler.createPreprocessor();
  compiler.createASTContext();
  compiler.setASTConsumer(new consumer(argv[argc - 1]));
  compiler.createSema(false, nullptr);
  (続)
```

InitializeBuiltins

```
  (続)
  auto& pp = compiler.getPreprocessor();
  pp.getBuiltinInfo().InitializeBuiltins(
      pp.getIdentifierTable(), pp.getLangOptions());

  auto& inputs =                                    .Inputs;
  if (inputs.siz
    compiler.Ini                                    [0].second);
    clang::Parse
      pp,
      &compiler.getASTConsumer(),
      compiler.getASTContext()
    );
  }
  return 0;
}
```

- __builtin_strcmp
- __builtin_sin
- __builtin_cos
- __builtin_sqrt …

clang::ParseAST

(続)

```
auto& pp = compiler.getPreprocessor();
pp.getBuiltinInfo().InitializeBuiltins(
    pp.getIdentifierTable(), pp.getLangOptions());

auto& inputs = compiler.getFrontendOpts().Inputs;
if (inputs.size() > 0) {
  compiler.InitializeSourceManager(inputs[0].second);
  clang::ParseAST(
    pp,
    &compiler.getASTConsumer(),
    compiler.getASTContext()
  );
}
retur
}
```

読込 →PP→ 解析 (AST)→AST Consumer

consumer

```
struct consumer : clang::ASTConsumer {
  explicit
  consumer(std::string const& target);
  virtual
  void Initialize(clang::ASTContext& ctx) /*override*/;
  virtual
  void HandleTopLevelDecl(clang::DeclGroupRef decls) /*override*/;
private:
  void handle_functiondecl(clang::FunctionDecl const* fd) const;

  std::string target_;
  clang::ASTContext* ctx_;
};
```

boost::optional<T&>のオーバヘッドが小さければなぁ

consumer::consumer, Initialize

```
struct consumer : clang::ASTConsumer {
  explicit
  consumer(std::string const& target)
  : target_(target), ctx_(nullptr) {
  }
  virtual
  void Initialize(clang::ASTContext& ctx) /*override*/ {
    ctx_ = &ctx;
  }
  virtual
  void HandleTopLevelDecl(clang::DeclGroupRef decls) /*override*/;
private:
  std::string target_;
  clang::ASTContext* ctx_;
};
```

consumer::HandleTopLevelDecl

```
struct consumer : clang::ASTConsumer {
  explicit
  consumer(std::string const& target);
  virtual
  void Initialize(clang::ASTContext& ctx) /*override*/;
  virtual
  void HandleTopLevelDecl(clang::DeclGroupRef decls) {
    for (auto& decl : decls)
      if (auto const* fd =
          llvm::dyn_cast<clang::FunctionDecl>(decl))
        handle_functiondecl(fd);
  }
private:
  void handle_functiondecl(clang::FunctionDecl const* fd) const;

  std::string target_;
  clang::ASTContext* ctx_;
};
```

宣言, clang/AST/Decl*.h

- Decl
- EnumDecl
- FunctionDecl
- TempladeDecl
- TypeDecl ...

文, clang/AST/Stmt*.h

- Stmt
- IfStmt
- DoStmt
- ForStmt
- ReturnStmt ...

式, clang/AST/Expr*.h

- Expr
- IntegerLiteralExpr
- BinaryOperator
- CallExpr
- InitListExpr ...

consumer::handle_functiondecl

```
struct consumer : clang::ASTConsumer {
  explicit
  consumer(std::string const& target);
  virtual
  void Initialize(clang::ASTContext& ctx) /*override*/;
  virtual
  void HandleTopLevelDecl(clang::DeclGroupRef decls) /*override*/;
private:
  void handle_functiondecl(clang::FunctionDecl const* fd) const {
    if (fd->getDeclName().getAsString() == target_) {
      auto const& range = fd->getSourceRange();
      range.getBegin().print(
        llvm::outs(), ctx_->getSourceManager());
      llvm::outs() << '¥n';
    }
  }
  std::string target_;
  clang::ASTContext* ctx_;
};
```

Eratta

入力ソースにエラーがあると… 落ちます.

error

```
in.cpp:8:3: error: use of undeclared identifier 'a'
      Stack dump:
      0. in.cpp:8:4: current parser token ';'
      1. in.cpp:6:10: parsing function body 'g'
      2. in.cpp:6:10: in compound statement ('')
      zsh: segmentation fault  ./where in.cpp f
```

Eratta

名前空間を解釈できません.

in2.cpp

```
namespace ns {
  void f() {}
}
```

shell

```
$ ./where in2.cpp f
$ ./where in2.cpp ns::f
```

→ clang::NamespaceDecl を再帰的に処理しないといけない.

組込み **AST Consumer**

- clang/Frontent/ASTConsumers.h
  - CreateASTPrinter
  - CreateASTDumper
  - CreateASTDumperXML
  - CreateASTViewer
  - CreateDeclContextPrinter

手元の環境だと CreateASTDumperXML が動かなかった…

clang::CreateASTDumper()

```
int f(int x, int y) (CompoundStmt 0x7f87e4917b00
                     <in.cpp:2:21, line:4:1>
 (ReturnStmt 0x7f87e4917ae0 <line:3:3, col:14>
  (BinaryOperator 0x7f87e4917ab8 <col:10, col:14> 'int' '+'
   (ImplicitCastExpr 0x7f87e4917a88 <col:10>
    'int' <LValueToRValue>
    (DeclRefExpr 0x7f87e4917a38 <col:10> 'int'
     lvalue ParmVar 0x7f87e49178c0 'x' 'int'))
    (ImplicitCastExpr 0x7f87e4917aa0 <col:14>
     'int' <LValueToRValue>
     (DeclRefExpr 0x7f87e4917a60 <col:14> 'int'
      lvalue ParmVar 0x7f87e4917920 'y' 'int')))))
```

Any Questions?