Clang-tidy: путешествие внутрь AST C++

Юрий Ефимочев

LOGICnow™

О себе

Архитектор в LogicNow
Специализация: высоконагруженные отказоустойчивые системы на C++

Бэкап-решение

IASO

MAXBackup™

Что такое clang-tidy?

Инструмент для статического анализа кода и поиска типичных ошибок программирования

Встроенные правила(~200)

ClangAnalyzer

Readability/Modernize/Performance

CppCoreGuidelines

LLVM/Google-style

Пример запуска

```
class TestClass
{
public:
    TestClass() :
        m_B(),
        m_C()
    {
    }

private:
    int m_A;
    int m_B;
    int m_C;
};
```

Пример запуска

```
$ clang-tidy -checks="cppcoreguidelines-*" ./test.cpp

1 warning generated.

/home/yury/Projects/test.cpp:4:5: warning: constructor does not initialize
these fields: m_A [cppcoreguidelines-pro-type-member-init]
        TestClass() :
        ^
        m_A(),
```

Code review

Цели code review

Соответствие реализации задаче

Поиск ошибок и неоптимальностей реализации

Соответствие *guidelines* и *best practices*

Clang-tidy и code review?

Платформа для построения собственных инструментов статического анализа кода

Расширяемость

Полный доступ к AST и препроцессору

Модульная архитектура

Инфраструктура для разработки и тестирования

Что необходимо для использования?

Код должен собираться clang

Необходима *compilation database*

Алгоритм разработки правила

1. Подготовить пример
2. Посмотреть в AST
3. ???
4. Profit

Пример: class field visibility

```
#include <iostream>

class TestClass
{
public:
    static int const Constant = 42;

    int m_public;   // style: class public field

private:
    int m_private;
};

struct TestStruct
{
    int Field;
};
```

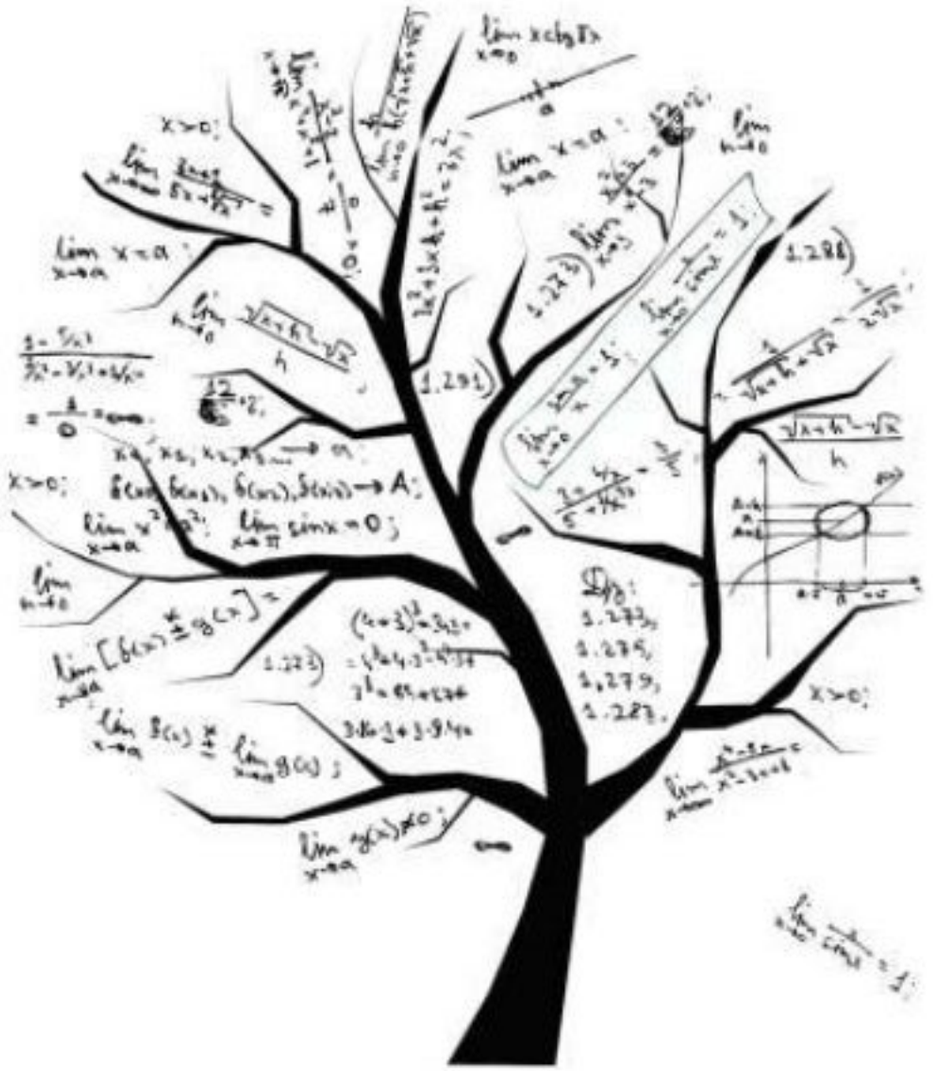
Clang-check: визуализация AST

```
$ clang-check -ast-dump -ast-dump-filter="Test" ./test.cpp

Dumping TestClass:
CXXRecordDecl 0x4ed7688 <test.cpp:3:1, line:12:1> line:3:7 class TestClass definition
|-CXXRecordDecl 0x4ed77a0 <col:1, col:7> col:7 implicit class TestClass
|-AccessSpecDecl 0x4ed7830 <line:5:1, col:7> col:1 public
|-VarDecl 0x4ed7868 <line:6:5, col:33> col:22 Constant 'const int' static cinit
| `-IntegerLiteral 0x4ed78c8 <col:33> 'int' 42
|-FieldDecl 0x4ed7940 <line:8:5, col:9> col:9 m_public 'int'
|-AccessSpecDecl 0x4ed7988 <line:10:1, col:8> col:1 private
`-FieldDecl 0x4ed79c0 <line:11:5, col:9> col:9 m_private 'int'

Dumping TestStruct:
CXXRecordDecl 0x4ed7a08 <test.cpp:14:1, line:17:1> line:14:8 struct TestStruct definition
|-CXXRecordDecl 0x4ed7b20 <col:1, col:8> col:8 implicit struct TestStruct
`-FieldDecl 0x4ed7bc0 <line:16:5, col:9> col:9 Field 'int'
```

Из чего состоит AST?



AST Nodes

Decl

Stmt

Type

AST Nodes

Decl

- CXXRecordDecl
- CXXMethodDecl
- VarDecl

Stmt

Type

AST Nodes

Decl

Stmt

- ifStmt
- CXXTryStmt
- BinaryOperator
- CompoundStmt

Type

AST Nodes

Decl

Stmt

Type

- PointerType
- ReferenceType
- LValueReferenceType
- RValueReferenceType

Clang-check: визуализация AST

```
$ clang-check -ast-dump -ast-dump-filter="Test" ./test.cpp

Dumping TestClass:
CXXRecordDecl 0x4ed7688 <test.cpp:3:1, line:12:1> line:3:7 class TestClass definition
|-CXXRecordDecl 0x4ed77a0 <col:1, col:7> col:7 implicit class TestClass
|-AccessSpecDecl 0x4ed7830 <line:5:1, col:7> col:1 public
|-VarDecl 0x4ed7868 <line:6:5, col:33> col:22 Constant 'const int' static cinit
| `-IntegerLiteral 0x4ed78c8 <col:33> 'int' 42
|-FieldDecl 0x4ed7940 <line:8:5, col:9> col:9 m_public 'int'
|-AccessSpecDecl 0x4ed7988 <line:10:1, col:8> col:1 private
`-FieldDecl 0x4ed79c0 <line:11:5, col:9> col:9 m_private 'int'

Dumping TestStruct:
CXXRecordDecl 0x4ed7a08 <test.cpp:14:1, line:17:1> line:14:8 struct TestStruct definition
|-CXXRecordDecl 0x4ed7b20 <col:1, col:8> col:8 implicit struct TestStruct
`-FieldDecl 0x4ed7bc0 <line:16:5, col:9> col:9 Field 'int'
```

AST Matchers

Node Matchers

cxxRecordDecl, cxxMethodDecl, namespaceDecl, ifStmt, ...

Narrowing Matchers

isConstant, isFinal, hasName, matchesName, unless, ...

Traversal Matchers

hasDescendant, hasParent, hasBody, ...

```
cxxMethodDecl(matchesName("Get.*"),
    hasParent(cxxRecordDecl(isStruct())))
```

Clang-query: поиск в AST

```
clang-query> match fieldDecl()

Match #1:
/home/yury/Projects/test.cpp:8:5: note: "root" binds here
    int m_public;
    ^~~~~~~~~~~~

Match #2:
/home/yury/Projects/test.cpp:11:5: note: "root" binds here
    int m_private;
    ^~~~~~~~~~~~~

Match #3:
/home/yury/Projects/test.cpp:16:5: note: "root" binds here
    int Field;
    ^~~~~~~~~
3 matches.
```

Clang-query: поиск в AST

```
clang-query> match fieldDecl(isPublic())

Match #1:

/home/yury/Projects/test.cpp:8:5: note: "root" binds here
    int m_public;
    ^~~~~~~~~~~~

Match #2:

/home/yury/Projects/test.cpp:16:5: note: "root" binds here
    int Field;
    ^~~~~~~~~
2 matches.
```

Clang-query: поиск в AST

```
clang-query> match fieldDecl(isPublic(), hasParent(cxxRecordDecl(isClass())))

Match #1:

/home/yury/Projects/test.cpp:8:5: note: "root" binds here
    int m_public;
    ^~~~~~~~~~~~
1 match.
```

Добавление правила

```
$ ./add_new_check.py misc field-visibility

Updating ./misc/CMakeLists.txt...
Creating ./misc/FieldVisibilityCheck.h...
Creating ./misc/FieldVisibilityCheck.cpp...
Updating ./misc/MiscTidyModule.cpp...
Creating ../test/clang-tidy/misc-field-visibility.cpp...
Creating ../docs/clang-tidy/checks/misc-field-visibility.rst...
Updating ../docs/clang-tidy/checks/list.rst...
Done. Now it's your turn!
```

Реализация правила

```
class FieldVisibilityCheck : public ClangTidyCheck
{
public:
    FieldVisibilityCheck(StringRef name, ClangTidyContext* context) :
        ClangTidyCheck(name, context)
    {
    }

private:
    void registerMatchers(ast_matchers::MatchFinder* finder) override
    {
    }

    void check(ast_matchers::MatchFinder::MatchResult const& result) override
    {
    }
};
```

Реализация правила

```
class FieldVisibilityCheck : public ClangTidyCheck
{
public:
    FieldVisibilityCheck(StringRef name, ClangTidyContext* context) :
        ClangTidyCheck(name, context)
    {
    }

private:
    void registerMatchers(ast_matchers::MatchFinder* finder) override
    {
        finder->addMatcher(fieldDecl(isPublic(), hasParent(cxxRecordDecl(isClass()))).bind("field"), this);
    }

    void check(ast_matchers::MatchFinder::MatchResult const& result) override
    {
        FieldDecl const& field = *result.Nodes.getNodeAs<FieldDecl const>("field");
        diag(field.getLocStart(), "Class field should be private");
    }
};
```

Что получилось?

```
$ clang-tidy -checks="misc-field-visibility-check" ./test.cpp

1 warning generated.
/home/yury/Projects/test.cpp:8:5: warning: Class field should be private
[iaso-field-visibility-check]
        int m_public;
        ^
```

Пример: argument immutability

```
int Function(int a)
{
    // ...

    a = something;

    // ...

    return a;
}
```

Пример: argument immutability

```
struct Test
{
    virtual int VirtualMethod(int a) = 0;

    int Method(int a, int const b, int& c, int* d)
    {
        a = b;
        return a;
    }
};
```

AST

```
Dumping Test:
CXXRecordDecl 0x46b1b20 <test.cpp:1:1, line:9:1> line:1:8 struct Test definition
|-CXXRecordDecl 0x46b1c30 <col:1, col:8> col:8 implicit struct Test
|-CXXMethodDecl 0x46b1d90 <line:3:5, col:32> col:9 VirtualMethod 'int (int)'
| `-ParmVarDecl 0x46b1cd0 <col:23, col:27> col:27 a 'int'
`-CXXMethodDecl 0x46b2140 <line:4:5, line:8:5> line:4:9 Method 'int (int, const int, int &, int *)'
  |-ParmVarDecl 0x46b1e50 <col:16, col:20> col:20 used a 'int'
  |-ParmVarDecl 0x46b1ec0 <col:23, col:33> col:33 used b 'const int'
  |-ParmVarDecl 0x46b1f60 <col:36, col:41> col:41 c 'int &'
  |-ParmVarDecl 0x46b2000 <col:44, col:49> col:49 d 'int *'
  `-CompoundStmt 0x46b2320 <line:5:5, line:8:5>
    |-BinaryOperator 0x46b22a0 <line:6:9, col:13> 'int' lvalue '='
    | |-DeclRefExpr 0x46b2238 <col:9> 'int' lvalue ParmVar 0x46b1e50 'a' 'int'
    | `-ImplicitCastExpr 0x46b2288 <col:13> 'int' <LValueToRValue>
    |   `-DeclRefExpr 0x46b2260 <col:13> 'const int' lvalue ParmVar 0x46b1ec0 'b' 'const int'
    `-ReturnStmt 0x46b2308 <line:7:9, col:16>
      `-ImplicitCastExpr 0x46b22f0 <col:16> 'int' <LValueToRValue>
        `-DeclRefExpr 0x46b22c8 <col:16> 'int' lvalue ParmVar 0x46b1e50 'a' 'int'
```

Реализация

```
void ImmutableParamsCheck::registerMatchers(MatchFinder* finder)
{
    finder->addMatcher(
        parmVarDecl(
            hasAncestor(functionDecl(hasBody(stmt()))),
            unless(anyOf(
                    hasType(isConstQualified()),
                    hasType(referenceType()),
                    hasType(pointerType())))).bind("parameter"), this);
}

void ImmutableParamsCheck::check(MatchFinder::MatchResult const& result)
{
    ParmVarDecl const& parameter = *result.Nodes.getNodeAs<ParmVarDecl const>("parameter");
    SourceLocation const location = parameter.getSourceRange().getEnd();

    diag(location, "Consider making constant") <<
        FixItHint::CreateInsertion(location, "const ");
}
```

Что получилось?

```
$ clang-tidy -checks="misc-immutable-parameters-check" -fix ./test.cpp

2 warnings generated.
/home/yury/Projects/test.cpp:4:20: warning: Consider making constant [iaso-immutable-
params]
          int Method(int a, int const b, int& c, int* d)
                     ^
                     const

/home/yury/Projects/test.cpp:4:20: note: FIX-IT applied suggested code changes
          int Method(int a, int const b, int& c, int* d)
                     ^

clang-tidy applied 1 of 1 suggested fixes.
```

Результат

```
struct Test
{
    virtual int VirtualMethod(int a) = 0;

    int Method(int a, int const b, int& c, int* d)
    {
        a = b;
        return a;
    }
};
```

Результат

```
struct Test
{
    virtual int VirtualMethod(int a) = 0;

    int Method(int const a, int const b, int& c, int* d)
    {
        a = b;
        return a;
    }
};
```

Пример: naming guidelines

```
// cxxRecordDecl(unless(matchesName("::[A-Z][a-zA-Z0-9]*$")))
class TestClass
{
public:
    // cxxMethodDecl(unless(matchesName("::[A-Z][a-zA-Z0-9]*$")))
    // parmVarDecl(unless(matchesName("::[a-z][a-zA-Z0-9]*$")))
    void Method(int arg);

private:
    // fieldDecl(unless(matchesName("::m_[a-z][a-zA-Z0-9]*$")),
    //     hasParent(cxxRecordDecl(isClass())))
    int m_field;
};

struct TestStruct
{
    // fieldDecl(unless(matchesName("::[A-Z][a-zA-Z0-9]*$")),
    //     hasParent(cxxRecordDecl(isStruct())))
    int Field;
};
```

Пример: naming guidelines

```
// varDecl(hasLocalStorage(), unless(matchesName("::[a-z][a-zA-Z0-9]*$")))
int localVariable;

// varDecl(hasGlobalStorage(),
//   unless(anyOf(matchesName("::s_[a-z][a-zA-Z0-9]*$"), hasType(isConstQualified()))))
static int s_staticVariable;

// varDecl(hasGlobalStorage(),
//   unless(anyOf(matchesName("::[A-Z][a-zA-Z0-9]*$"),
unless(hasType(isConstQualified()))))
static int const Constant = 42;
```

Clang-tidy

Clang-tidy: итоги

Простая реализация сложных проверок

Атоматизизация рутинных проверок

Отличный способ лучше узнать C++

Полезные ссылки

http://clang.llvm.org/extra/clang-tidy

http://clang.llvm.org/docs/LibASTMatchersReference.html

http://clang.llvm.org/docs/IntroductionToTheClangAST.html

efimyury@gmail.com
yury.efimochev@logicnow.com

LOGICnow™