#C3Days 2021

# Clang: автоматизация работы с кодом

Александр Тулуп
tulup@ascon.ru

# Цель

- переход на linux

# Что есть ?

- ~3 миллиона строк legacy кода под msvc

АСКОН

# Что нужно ?

- типовые правки, много
- нужна информация о коде
- нестандартные проверки
- генерация кода

# Пример

```
// std::auto_ptr нет в c++17, но для msvc _HAS_AUTO_PTR_ETC
std::vector<std::auto_ptr<MbNurbs>> curves;
std::auto_ptr<MbNurbs> curve = get_data();
curves.push_back( curve );
```

# grep

```
// replace std::auto_ptr -> std::unique_ptr
std::vector<std::unique_ptr<MbNurbs>> curves;
std::unique_ptr<MbNurbs> curve = get_data();
curves.push_back( curve ); // std::move
```

# Готовые инструменты ?

# clang-tidy

```
// один файл
clang-tidy -checks=-*,modernize-replace-auto-ptr -fix foo.cpp

// весь проект, параллельно
run-clang-tidy.py -checks=-*,modernize-replace-auto-ptr -fix
```

```
#include <utility> // add include

std::vector<std::unique_ptr<MbNurbs>> curves;
std::unique_ptr<MbNurbs> curve = get_data();
curves.push_back( std::move(curve) ); // add std::move
```

# clang-tidy

- modernize-replace-auto-ptr
- modernize-use-override
- google-readability-casting

# clang-tidy

- https://clang.llvm.org/extra/clang-tidy/checks/list.html

```
clang-tidy --checks=* --list-checks
```

# Информация о коде

# clang-query

```
int main()
{
  int id = 42;
  auto * name = "str";
}
```

# AST

```
FunctionDecl 0x1361e90 </mnt/dev/_/cpp/main.cpp:6:1, line:10:1> l
`-CompoundStmt 0x1362398 <line:7:1, line:10:1>
  |-DeclStmt 0x1362040 <line:8:3, col:14>
  | `-VarDecl 0x1361fb8 <col:3, col:12> col:7 id 'int' cinit
  |   `-IntegerLiteral 0x1362020 <col:12> 'int' 42
  `-DeclStmt 0x1362380 <line:9:3, col:22>
    `-VarDecl 0x13620f8 <col:3, col:17> col:10 name 'const char *
      `-ImplicitCastExpr 0x1362368 <col:17> 'const char *' <Array
        `-StringLiteral 0x13621d8 <col:17> 'const char [4]' lvalu
```

# clang-query: примеры

```
FunctionDecl 0x1361e90 </mnt/dev/_/cpp/main.cpp:6:1, line:10:1> l
`-CompoundStmt 0x1362398 <line:7:1, line:10:1>
  |-DeclStmt 0x1362040 <line:8:3, col:14>
  | `-VarDecl 0x1361fb8 <col:3, col:12> col:7 id 'int' cinit
  |   `-IntegerLiteral 0x1362020 <col:12> 'int' 42
  `-DeclStmt 0x1362380 <line:9:3, col:22>
    `-VarDecl 0x13620f8 <col:3, col:17> col:10 name 'const char *
      `-ImplicitCastExpr 0x1362368 <col:17> 'const char *' <Array
        `-StringLiteral 0x13621d8 <col:17> 'const char [4]' lvalu
```

```
set output detailed-ast
// переменные с именем id
match declStmt(has(varDecl( hasName("id") ).bind("x")))
```

```
VarDecl 0x13620f8 </mnt/dev/_/cpp/main.cpp:9:3, col:17> col:10 na
`-ImplicitCastExpr 0x1362368 <col:17> 'const char *' <ArrayToPoin
  `-StringLiteral 0x13621d8 <col:17> 'const char [4]' lvalue "str
```

# clang-query: примеры

```
class A {};
class B {};
class C: public A, public B {};
class D: public A {};
```

```
set output diag
// наследники A но не B
match cxxRecordDecl(
      isClass(), isDerivedFrom("A"), unless(isDerivedFrom("B"))
    )
```

```
class D: public A {};
^~~~~~~~~~~~~~~~~~~~
```

# Свои проверки

# llvm api

- clang api C/C++

АСКОН

# llvm C++ api

- достоинства
  - доступны все возможности компилятора
  - AstMatchers ( код можно брать из clang-query )
  - можно написать plugin для clang
- недостатки
  - меняется с каждой версией
  - нужен llvm целиком

# llvm C api

- достоинства
  - стабильное api
  - есть реализация на python (*pip install clang*)
  - удобен для написания скриптов
- недостатки
  - ограничен функционал
  - для обхода ast используется visiter

# Что еще ?

# Include What You Use (iwyu)

- удаляет лишние include
- добавляет forward declaration
- полезно для ускорения сборки
- уменьшение связности кода

# Include What You Use (iwyu)

```
// найти лишние include
iwyu_tool -j 256 -p compile_commands.json -- -w > iwyu_res.cpp

~5000 файлов, ~8 минут
```

```
// применить исправления
fix_includes.py < iwyu_res.cpp
```

# compile_commands.json

```
cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=TRUE
```

Спасибо!

# Ссылки

- Adding a New clang-tidy Check by the Practice
- ClangShar
- Clang-Tidy для автоматического рефакторинга кода
- Extra Clang Tools
- Include What You Use
- Implementing Include-what-you-use Using Clang
- How to build a C++ processing tool using the Clang
- Пример разбора C++ кода с помощью libclang на Python