Extending Clang AST Matchers to Preprocessor Constructs

Jeff Trull

3 November 2016

Motivation

- I had just learned all about libTooling
- I had a friend with a legacy C-style codebase

```
// lots of conditional compilation:
#ifdef SIMULATION
// update event counters
// trigger state changes
// propagate simulation results
#else
// read hardware registers
// log things on console
// write hardware registers
// busy loop sleep
#endif
```

I unwisely claimed I could automatically refactor this for him

Concrete Example

QString vs. std::string

```
char const* cstr = "foo";
#ifdef USE_QSTRING
    using string_t = QString;
    string_t s(cstr);
    s = s.toUpper();
#else
    using string_t = std::string;
    string_t s(cstr);
    std::transform(s.begin(), s.end(), s.begin(),
                   [](char c) { return std::toupper(c); });
#endif
```

Concrete Example

QString vs. std::string

```
char const* cstr = "foo";
#ifdef USE_QSTRING
    using string_t = QString;
    string_t s(cstr);
    s = s.toUpper();
#else
    using string_t = std::string;
    string_t s(cstr);
    std::transform(s.begin(), s.end(), s.begin(),
                   [](char c) { return std::toupper(c); });
#endif
```

Concrete Example

QString vs. std::string

```
char const* cstr = "foo";
#ifdef USE_QSTRING
    using string_t = QString;
    string_t s(cstr);
    s = s.toUpper();
#else
    using string_t = std::string;
    string_t s(cstr);
    std::transform(s.begin(), s.end(), s.begin(),
                   [](char c) { return std::toupper(c); });
#endif
```

Concrete Example

Class templated on PP condition

```
template<bool UsingQString>
struct StringClass {
    // base template handles true case
    using string_t = QString;
    static void to_upper(string_t& s) {
        s = s.toUpper();
    }
};

template<>
struct StringClass<false> {
    using string_t = std::string;
    static void to_upper(string_t& s) {
        std::transform(s.begin(), s.end(), s.begin(),
                       [](char c) { return std::toupper(c); });
    }
};
```

Concrete Example

Class templated on PP condition

```
template<bool UsingQString>
struct StringClass {
    // base template handles true case
    using string_t = QString;
    static void to_upper(string_t& s) {
        s = s.toUpper();
    }
};

template<>
struct StringClass<false> {
    using string_t = std::string;
    static void to_upper(string_t& s) {
        std::transform(s.begin(), s.end(), s.begin(),
                       [](char c) { return std::toupper(c); });
    }
};
```

Concrete Example

Class templated on PP condition

```
template<bool UsingQString>
struct StringClass {
    // base template handles true case
    using string_t = QString;
    static void to_upper(string_t& s) {
        s = s.toUpper();
    }
};

template<>
struct StringClass<false> {
    using string_t = std::string;
    static void to_upper(string_t& s) {
        std::transform(s.begin(), s.end(), s.begin(),
                       [](char c) { return std::toupper(c); });
    }
};
```

Concrete Example

Class template usage

```
// select policy class in a single place
#ifdef USE_QSTRING
using StringPolicy = StringClass<true>;
#else
using StringPolicy = StringClass<false>;
#endif

void my_fn() {
    using string_t = StringPolicy::string_t;
    string_t s("foo");           // chooses appropriate type
    StringPolicy::to_upper(s);   // calls appropriate code
    ...
}
```

Concrete Example

Class template usage

```
// select policy class in a single place
#ifdef USE_QSTRING
using StringPolicy = StringClass<true>;
#else
using StringPolicy = StringClass<false>;
#endif

void my_fn() {
    using string_t = StringPolicy::string_t;
    string_t s("foo");           // chooses appropriate type
    StringPolicy::to_upper(s);   // calls appropriate code
    ...
}
```

Concrete Example

Class template usage

```
// select policy class in a single place
#ifdef USE_QSTRING
using StringPolicy = StringClass<true>;
#else
using StringPolicy = StringClass<false>;
#endif

void my_fn() {
    using string_t = StringPolicy::string_t;
    string_t s("foo");           // chooses appropriate type
    StringPolicy::to_upper(s);   // calls appropriate code
    ...
}
```

Tools

What does Clang provide that can help?

PPCallbacks

Hooking conditional ranges

```
struct PPActions : clang::PPCallbacks
{
    void Ifdef(clang::SourceLocation loc,
               clang::Token const& tok,
               clang::MacroDefinition const& md) override {
        // if this is our conditional, remember it
        ...
    }
    void Endif(clang::SourceLocation endifloc,
               clang::SourceLocation ifloc) override {
        // if this ends one of our conditionals, record SourceRange
        ...
    }
```

PPCallbacks

Hooking conditional ranges

```
struct PPActions : clang::PPCallbacks
{
    void Ifdef(clang::SourceLocation loc,
               clang::Token const& tok,
               clang::MacroDefinition const& md) override {
        // if this is our conditional, remember it
        ...
    }
    void Endif(clang::SourceLocation endifloc,
               clang::SourceLocation ifloc) override {
        // if this ends one of our conditionals, record SourceRange
        ...
    }
```

PPCallbacks

Hooking conditional ranges

```
struct PPActions : clang::PPCallbacks
{
    void Ifdef(clang::SourceLocation loc,
               clang::Token const& tok,
               clang::MacroDefinition const& md) override {
        // if this is our conditional, remember it
        ...
    }
    void Endif(clang::SourceLocation endifloc,
               clang::SourceLocation ifloc) override {
        // if this ends one of our conditionals, record SourceRange
        ...
    }
```

Custom AST Matcher

Matching conditional ranges to AST nodes


```
AST_MATCHER_P(clang::Stmt,                statementInRange,
              clang::SourceRange,         range) {
    clang::SourceManager const& sm =
        Finder->getASTContext().getSourceManager();
    // is the statement node entirely within the supplied range?
    return !sm.isBeforeInTranslationUnit(Node.getLocStart(),
                                         range.getBegin()) &&
           !sm.isBeforeInTranslationUnit(range.getEnd(),
                                         Node.getLocEnd());
}
```

Custom AST Matcher

Matching conditional ranges to AST nodes


```
AST_MATCHER_P(clang::Stmt,                    statementInRange,
              clang::SourceRange,             range) {
    clang::SourceManager const& sm =
        Finder->getASTContext().getSourceManager();
    // is the statement node entirely within the supplied range?
    return !sm.isBeforeInTranslationUnit(Node.getLocStart(),
                                         range.getBegin()) &&
           !sm.isBeforeInTranslationUnit(range.getEnd(),
                                         Node.getLocEnd());
}
```

Custom AST Matcher

Matching conditional ranges to AST nodes

```
AST_MATCHER_P(clang::Stmt,                statementInRange,
              clang::SourceRange,         range) {
    clang::SourceManager const& sm =
        Finder->getASTContext().getSourceManager();
    // is the statement node entirely within the supplied range?
    return !sm.isBeforeInTranslationUnit(Node.getLocStart(),
                                         range.getBegin()) &&
           !sm.isBeforeInTranslationUnit(range.getEnd(),
                                         Node.getLocEnd());
}
```

Lambda Hack

To create an AST scope for the conditional text

To analyze the code within the conditionals we can insert lambdas:

```
auto expression_capture_0 = [&]() -> void {  // inserted
    s = s.toUpper();
}                                            // inserted
expression_capture_0();                      // inserted
```

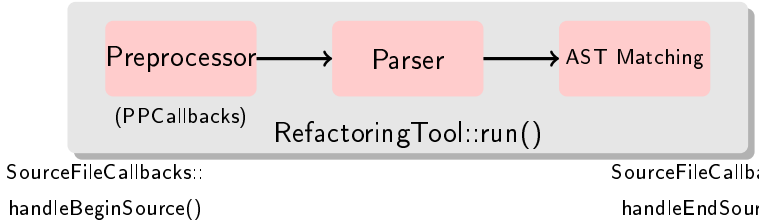
We run the analysis twice: once for the true case, once for false. This gives us:

- A set of statements
- Variables used or modified

from which we can create methods.

Necessary Hooks

The big picture has a flaw



- SourceFileCallbacks::handleBeginSource() allows us to install PPCallbacks
- SourceFileCallbacks::handleEndSource() gives access to AST and SourceManager
- We need to install matchers after the preprocessor and before matching

Diagnostic Hook
Connecting ranges to matchers

Hey, what's this?

clang 3.9.0svn

Main Page | Related Pages | Modules | Namespaces | Classes | Files

Class List | Class Index | Class Hierarchy | Class Members

clang › ast_matchers › MatchFinder › ParsingDoneTestCallback

clang::ast_matchers::MatchFinder::ParsingDoneTestCallback Class Reference abstract

Called when parsing is finished. Intended for testing only. More...

```
#include <ASTMatchFinder.h>
```

Public Member Functions

virtual	~ParsingDoneTestCallback ()
virtual void	run ()=0

Better Support

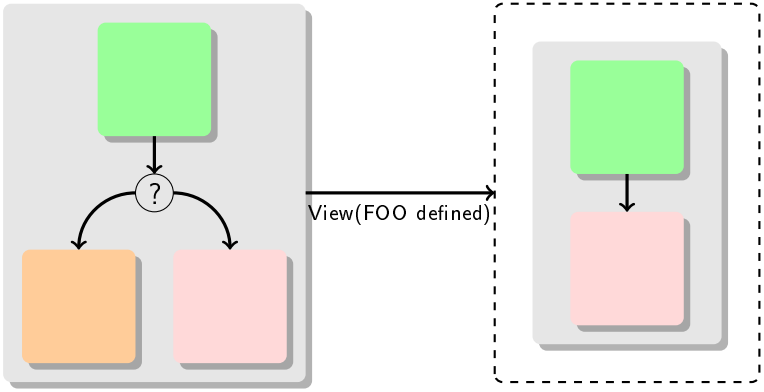
- Make ParsingDoneTestCallback official
  - Someone else is going to find this useful
- Incorporate preprocessor conditions into the AST
  - Previously used in academic research
  - Tricky...

Incorporating the Preprocessor

- Add a custom attribute with conditional info
  - Easy and backward compatible
  - Can only handle one configuration
- Introduce a conditional AST node
  - Add a "view" wrapper on configurations [1] to preserve compatibility
  - Can start with just the one parsed configuration
  - Problem: conditionals can change the context and thus semantics of other code
    - usually this is fairly perverse code

[1] sets of macro definitions

Incorporating the Preprocessor
Conditional nodes with views



Followup

- Interested? Please seek me out
- https://github.com/jefftrull/octothorpe