Attributes reflection in Clang

A first time contributor experience

Implementation

Aurelien Cassagnes - **Bloomberg LP, Tokyo**

C++ reflection in 26 · A quickest digest

Reflection

GRAMMAR

reflect-expression:

^^ ::
^^ *unqualified-id*
^^ *qualified-id*
^^ *type-id*
^^ *pack-index-expression*
^^ *[[attribute]]*

CODE

```
constexpr std::meta::info r = ^^[[nodiscard ]];
```

- Tagged opaque type in *APValue, here a ParsedAttr**
- Heavy lifting done via *MaybeParseCXX11Attributes*

Splice

GRAMMAR

attribute-specifier:

[[attribute-using-prefix$_{opt}$ *attribute-list]]*
[[splice-specifier]]

splice-specifier:

[: constant-expression :]

CODE

```
enum class [[nodiscard]] ErrorCode {
  warn,
  fatal,
};

enum class [[ [: ^^ErrorCode :] ]] ClosedErrorCode {
```

- Hook into *MaybeParseCXX11Attributes* on splice token
- Evaluate *constant-expression* if possible, and reinject an annotated token

Clang implementation · Attribute

Attribute parsing digest

```
enum class [[nodiscard]] E { /* … */ };
```

1. Parse declaration specifier
2. Parse attribute token and arguments
3. Build a generic ParsedAttr and pass it to Sema
4. Giant switch on attribute type, create a specific Attr
5. Attach Attr to node in AST, ParsedAttr goes away

attr.td

[ClangAttrEmitter.cpp] ← [attr.td] → [Attrs.inc]

```
def WarnUnusedResult : InheritableAttr {
  let Spellings = [
                  CXX11<"", "nodiscard", 201907>,
                  /* … /*,
                  GCC<"warn_unused_result">];
  let Subjects = SubjectList<[
                  ObjCMethod, Enum, Record, FunctionLike, TypedefName]>;
  let Args = [StringArgument<"Message", 1>];
  let Documentation = [WarnUnusedResultsDocs];
  let AdditionalMembers = [{
    // Check whether this the C++11 nodiscard version, even in non C++11
    // spellings.
    bool IsCXX11NoDiscard() const {
      return this->getSemanticSpelling() == CXX11_nodiscard;
    }
  }];
}
```

Clang implementation · Attribute arguments

Syntactic / Semantic attribute

Sample

```
enum class [[nodiscard("yup")]] Foo {};
constexpr auto noDiscard = attributes_of(^^Foo)[0];
```

- An *Attr* is attached to *Foo* type declaration node
- *attributes_of()* looks up the latest declaration and fetching all attached *Attr*

What's the problem

To build a reflection you want a ParsedAttr
Looking into the AST only gives you Attr

Hack #1

1. Extend ParsedAttr lifetime to outlive declaration parsing
2. Make the final Attr own the raw ParsedAttr from (1)
3. Navigate the chain of links
   Decl → Attr → ParsedAttr → Args

Takeaway

1. Navigating where **Sema** starts and **Parser** end is complex but you get used.
2. It is critical to understand the codegen contraptions
3. Still not convinced we don't want a generic **getArgs()** on **Attr**

Clang implementation · In-place dependent splicing

Dependent expression

Classic

```
template <class T>
auto Foo() {
    return sizeof(T) * 2;
};
```

Reflection

```
class [[maybe_unused]] Bar {};
// …
template <class T>
                class [[ [: ^^T :] ]] Foo { /* … */ };
Foo<Bar> s;  // Should be decorated with [[maybe_unused]]
```

What's the problem

- When parsing Foo template, we must make note that some attributes may be attached later
- Evaluating that expression, fetching the attributes need to happen at instantiation

Hack #2

1. Add a DelayedSplice custom attribute to supported set
2. When parsing *[[[:* synthesize a DelayedSplice attribue and stash the expression inside
3. Hook into Sema::InstantiateAttr() to evaluate the stashed expression

Takeaway

1. Other people *likely* had the same issues that you have, it may already be solved.
2. Understanding the codegen system ***really*** pays off
3. Understanding tree transform is intimidating enough to come up with hack like this…

Thank you

Aurelien Cassagnes - **Bloomberg LP, Tokyo**