Attributes introspection

Document #:
Date: 2024-04-20
Project: Programming Language C++
Audience: sg7
Reply-to: Aurelien Cassagnes
<aurelien.cassagnes@gmail.com>

Contents

1 Introduction

The current draft aims at getting the conversation and work started on supporting introspection of attributes. There is ongoing work to refine that draft, especially when it comes to motivating examples.

2 Motivation

Attributes are used to great extent and there likely will be attributes added as the language evolve. What is missing now is a way for generic code to look into the attributes related to an entity. A motivating example is the following

```
[[nodiscard]] bool foo(int i) { return i % 2 ; }

template <class F, class... Args>
constexpr std::invoke_result_t<F, Args...> logInvoke(F&& f, Args&&... args) {
    // Do some extra work, log the call...
    // Fwd call
    return std::invoke(std::forward<F>(f), std::forward<Args>(args)...);
}

int main() {
    foo(0); // Warning on discarded return
    logInvoke(foo, 0); // No warning on discarded return
}
```

Other examples of wrapping around callables can be found, whether by closure or explicity registering callbacks for dispatch, etc. The current example deals with code as it is in a c++23 world, but other applications can easily be thought of in the context of code injection [Metaprogramming] where one may want to skip over `[[deprecated]]` members for example.

3 Proposal

We put ourselves in the context of [Reflection] for the current proposal to be more illustrative in terms of what is being proposed.

3.1 Reflection operator

If our understanding is correct, the proposition for `^` grammar does not cover attributes , as in `^[[deprecated]]` is meaningless. We think this will limit the potential use of attributes introspection. The current proposal advocates for

```
^ attribute
```

to be well formed.

3.2 Splicers

We propose that the form

```
attribute [: r :]
```

be supported. This implicitly means that `std::meta::info` must be expanded, this will be discussed thereafter.

— `attribute [: r :]` produces a potentially empty sequence of attributes corresponding to the attributes that are attached to `r`

3.3 std::meta::info

We propose that attributes be a supported *reflectable* property of the expression that are reflected upon. That means value of type `std::meta::info` should be able to represent an attribute in addition to the current supported set.

3.4 Metafunctions

We propose to add a metafunction to what is discussed already in [Reflection]

```
template<typename E>
  consteval auto attributes_of(E entity) -> vector<info>;
```

This being applied to an entity `E` will yield a sequence of `std::meta::info` representing the attributes attached to `E`.

3.5 Queries

We do not think it is necessary to introduce query or queries at this point. Especially we would not recommend to introduce a dedicated query per attribute (eg `is_nodiscard`, `is_nouniqueaddress`, etc.)

4 Discussion

Originally the idea of introducing a `declattr(Expression)` keyword seemed the most straightforward to tackle on this problem, but from feedback the concern of introspecting on expression attributes was a concern that belongs with the reflection SG. The current proposal shifted away from the original `declattr` idea to align better

with the reflection toolbox. Note also that as we advocate here for `attribute [: r :]` to be supported, we recover the ease of use that we first envisioned `declattr` to have.

Another item of discussion is whether vendor specific attributes should be supported or only the ones described by the standard, feedbacks on this is inexistent yet.

5 References

[Metaprogramming] Andrew Sutton. Metaprogramming.
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p2237r0.pdf

[Reflection] Wyatt Childers, Peter Dimov, Dan Katz, Barry Revzin, Andrew Sutton, Faisal Vali, and Daveed Vandevoorde. Reflection for c++26.
https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p2996r2.html