Proposal for attributes introspection

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

Contents

1 Introduction

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 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. Other applications can easily be thought of in the context of code injection [Metaprogramming] where one may want to skip over `[[deprecated]]` members

2 Proposal

We put ourselves in the context of [Reflection] for the proposal to be more illustrative.

2.1 std::meta::info

We propose that attributes be a supported *reflectable* property of the expression that are reflected upon. That means `std::meta::info` should be augmented to represent an attribute in addition to what it can represent.

2.2 Metafunctions

We propose to add a function 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 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.

4 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