Detecting in C++ whether a type is defined, part 1: The initial plunge

devblogs.microsoft.com/oldnewthing/20190708-00

July 8, 2019

Raymond Chen

Warning to those who got here via a search engine: Don't use this version. Keep reading to the end of the series.

Suppose you want to be able to detect in C++ whether a type has been defined. For example, maybe you want to use a type if it exists. This can happen if, say, you are a library like React Native for Windows, and you need to be able to run with different versions of the Windows SDK. Or you're writing a library where the client can customize the behavior by defining another class with a well-known name. Perhaps you're trying to mimic C# partial classes.

My initial idea was to take advantage of unqualified name lookup by creating an alternate definition for the type that sits at a lower priority than the one we're looking for.

```
// awesome.h
namespace awesome
{
  // might or might not contain
  struct special { ... };
}

// your code
namespace detect::impl
{
  struct not_implemented {};
  using special = not_implemented;
}

namespace awesome::detect
{
  using namespace ::detect::impl;
  constexpr bool is_special_defined =
    !std::is_same_v<special, ::detect::impl::not_implemented>;
}
```

The idea here is that I declare an alternate version of the `special` structure in the `detect::impl` namespace, and place it in the search order at a location that comes *after* searching in the `awesome` namespace.

The `using namespace ::detect ::impl;` directive makes the names from the `detect::impl` visible as if they had been declared in the global namespace. Why the global namespace? Because the rule for `using namespace` is that the names from the imported-from namespace are treated as if they had been declared in the namespace which is the nearest common ancestor of the importing namespace and the imported-from namespace. In our case, the imported-from namespace is `::detect ::impl` and the importing namespace is `::awesome ::detect`. Since they don't even share a common top-level namespace, the nearest common ancestor is the global namespace.

Next, I check the name `special`. The unqualified name lookup searches in the following order:

- `::awesome ::detect ::special`
- `::awesome ::special`
- `::special` (which, thanks to our `using namespace ::detect ::impl;` directive also searches in `::detect ::impl`.)

There is definitely no `special` declared in the `::awesome ::detect` namespace, so it comes down to the other two. If it exists in the `::awesome` namespace, then the unqualified lookup will find that type; otherwise, it will find the one in the `::detect ::impl` namespace.

We then use `std:: is_same_v` to see whether the type we found is our fake one.

This works, but it's awkward because you have to do the detection from inside the `::awesome ::detect` namespace, since that's where we set up the search order. For every type you want to detect, you need to create an alias in the `::detect ::impl` namespace and a custom `is_whatever_defined` constant.

Next time, we'll look at my second attempt.

Raymond Chen

Follow