Template metaprogramming trick: Get the compiler to tell you what type you have

devblogs.microsoft.com/oldnewthing/20200528-00

May 28, 2020

Raymond Chen

C++ template metaprogramming is like writing a program in Prolog without a debugger. The compiler executes your metaprogram by running through a bunch of pattern-matching rules. But unlike Prolog, the C++ template metaprogramming language doesn't have a debugger. You just feed your code to the compiler, and you get a few possible results:

1. It fails to compile.
2. It compiles and gives you what you want.
3. It compiles and gives you something that wasn't what you want.

Only if you're lucky do you get case 2 on the first try.

There's no way to single-step through your metaprogram, and there's no print-debugging either. All you can do is see what the compiler says.

Here's a trick I use to get *something*. It's not great, but it's still handy.

```
template<typename... Args> void whatis();
```

This is a forward declaration of a function that takes an arbitrary number of type arguments.

I can drop a call to this function at various points in my template metaprogram to see how the compiler deduced a type:

```
template<typename T>
void f(T&& t)
{
 whatis<T>();
 ... other stuff ...
}
```

When I instantiate `f`, a call to `whatis<T>` is made, among all the other stuff. I can look at the compiler output or the linker's "unresolved external" error message to see what `T` ended up being.

```
double v = 3.0;
f(v);

// msvc

??$f@AEAN@@YAXAEAN@Z PROC                  ; f<double &>, COMDAT
        ... other stuff ...
        call    ??$whatis@AEAN@@YAXXZ       ; whatis<double &>
        ... other stuff ...
??$f@AEAN@@YAXAEAN@Z ENDP                  ; f<double &>

unresolved external symbol
"void __cdecl whatis<double &>()" (??$whatis@AEAN@@YAXXZ)

// gcc
_Z1fIRdEvOT_:
        ... other stuff ...
        call    _Z6whatisIJRdEEvv
        ... other stuff ...

undefined reference to `void whatis<double&>()'
```

Aha, in this instantiation of `f`, the type `T` was deduced to be `double&`.

It's not a super-awesome trick, but with template metaprogramming, every little bit helps.

Raymond Chen

Follow