A brief introduction to C++ structured binding

devblogs.microsoft.com/oldnewthing/20201014-00

October 14, 2020

Raymond Chen

C++17 introduced a feature known as *structured binding*. It allows a single source object to be taken apart:

```
std::pair<int, double> p{ 42, 0.0 };
auto [i, d] = p;
// int i = 42;
// double d = 0.0;
```

It seems that no two languages agree on what to call this feature. C# calls it *deconstructing*. JavaScript calls it *destructuring*. (Python doesn't seem to have a specific name for this concept, although the common case where the source is a list does have the name *list comprehension*.) Python calls it *unpacking*. My guess is that C++ avoided both of these terms to avoid confusion with the word *destructor*.

There is a subtlety in the way structured binding works: Binding qualifiers on the `auto` apply to how the *source* is bound, not on how the *destination* is bound.[1]

For example,

```
auto&& [i, d] = p;
```

becomes (approximately)[1]

If `p.get<N>` exists	If `p.get<N>` does not exist

`auto&& hidden = p;`
`decltype(auto) i = p.get<0>();`
`decltype(auto) d = p.get<1>();`
	`auto&& hidden = p;`
	`decltype(auto) i = get<0>(p);`
	`decltype(auto) d = get<1>(p);`

where `hidden` is a hidden variable introduced by the compiler. The declarations of `i` and `d` are inferred from the `get` method or free function.[2]

(In a cruel twist of fate, if `hidden` is const or a const reference, then that const-ness propagates to the destinations. But the reference-ness of `hidden` does not propagate.)

The `decltype(auto)` means that the reference-ness of the return type is preserved rather than decayed. If `get` returns a reference, that reference or qualifier is preserved. This differs from `auto` which will decay references to copies.

All of this comes into play when you want to make your own objects available to structured binding, which we'll look at next time.

Bonus chatter: There is no way to specify that you want only selected pieces. You must bind all the pieces.

[1] In reality, the bound variables have underlying type `std::tuple_element_t<N, T>` (where `N` is the zero-based index and `T` is the type of the source), possibly with references added. But in practice, these types match the return types of `get`, so it's easier just to say that they come from `get`.

[2] My reading of the language specification is that the destination variables are always references:

> **[dcl.struct.bind]**
> 3. … [E]ach $\mathtt{v}_i$ is a variable of type "reference to $\mathtt{T}_i$" initialized with the initializer, where the reference is an lvalue reference if the initializer is an lvalue and an rvalue reference otherwise.

and therefore the expansion of the structured binding would be

```
auto&& i = get<0>(p);
auto&& d = get<1>(p);
```

However, in practice, the compilers declare the destination variables as matching the return value of `get`, as I noted above.

So I must be reading the specification wrong.

(The text was revised for C++20, but even in the revision, it's still a reference.)

[1] That's because a structured binding *really* is a hidden variable plus a bunch of references to the pieces of that hidden variable. That's why the qualifiers apply to the hidden variable, not to the aliases.

Raymond Chen

Follow