How can I choose a different C++ constructor at runtime?

devblogs.microsoft.com/oldnewthing/20250306-00

Zura For Reg March 6, 2025

Raymond Chen

Suppose you have a class with two constructors.

```
struct WidgetBase
{
    // local mode
    WidgetBase();

    // remote mode
    WidgetBase(std::string const& server);

    // The mutex makes this non-copyable, non-movable
    std::mutex m_mutex;
};

struct WidgetOptions
{
    ⟦ random stuff ⟧
};

struct Widget : WidgetBase
{
    Widget(WidgetOptions const& options) :
        // This doesn't work
        CanBeLocal(options)
            ? WidgetBase()
            : WidgetBase(GetServer(options))
    {}

    static bool CanBeLocal(WidgetOptions const&);
    static std::string GetServer(WidgetOptions const&);
};
```

We want to use the base class's local constructor if the options are compatible with a local Widget. Otherwise, we have to create a remote Widget. But you can't choose a base class constructor at runtime. Your constructor has to call the base class constructor somehow, and

by that point the decision has already been made.

You might try using the ternary operator.

```
    Widget(WidgetOptions const& options) :
        WidgetBase(
            CanBeLocal(options)
                ? WidgetBase()
                : WidgetBase(GetServer(options)))
    {}
```

But this doesn't work because it invokes the copy and/or move constructor: The ternary operator produces a `WidgetBase` by one means or another, and then we have to copy/move the temporary into the base class `WidgetBase` object.

The secret, once again, is to take advantage of copy elision.

```
struct Widget : WidgetBase
{
    Widget(WidgetOptions const& options) :
        WidgetBase(ChooseWidgetBase(options))
    {}

    static bool CanBeLocal(WidgetOptions const&);
    static std::string GetServer(WidgetOptions const&);

private:
    static WidgetBase ChooseWidgetBase(
        WidgetOptions const& options)
    {
        if (CanBeLocal(options)) {
            return WidgetBase();
        } else {
            return WidgetBase(GetServer(options));
        }
    }
};
```

This looks the same as the ternary, just moved out of line, but it's subtly different.

The difference is that all of the `return` statements use one of the magic copy elision forms: `return WidgetBase(⟦...⟧)`. This allows the compiler to construct the `WidgetBase` object directly into the return value, and when called from the `Widget` constructor, the return value is the `WidgetBase` base class object.

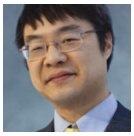
If you like throwing everything inline, you can use a lambda to put the helper directly into the base class constructor arguments.

```
Widget(WidgetOptions const& options) :
    WidgetBase([&] {
        if (CanBeLocal(options)) {
            return WidgetBase();
        } else {
            return WidgetBase(GetServer(options));
        }
    }())
{}
```

Bonus chatter: The problem with the ternary is that the ternary expression is not a copy elision candidate. The rule for ternary expressions is that the result is *initialized from* the branch of the ternary that is selected. The value from the branch is copied/moved into the expression result, and it is the result that is constructed in place.

Author

Raymond Chen



Raymond has been involved in the evolution of Windows for more than 30 years. In 2003, he began a Web site known as The Old New Thing which has grown in popularity far beyond his wildest imagination, a development which still gives him the heebie-jeebies. The Web site spawned a book, coincidentally also titled The Old New Thing (Addison Wesley 2007). He occasionally appears on the Windows Dev Docs Twitter account to tell stories which convey no useful information.

6 comments

Discussion is closed. Login to edit/delete existing comments.

Newest

- ZF

  March 10, 2025
  @Raymond Chen

  Off-topic: commenting here since the discussion is closed in some older posts about Windows 7 calculator. Could you please mention what UI API is used for “Worksheets > Mortgage” and similar panels? In the Win7 calc.exe which is a native Win32 app as I can tell. Thank you!

- 

  Matt McCutchen March 7, 2025 · Edited

  So as far as knows, it's constructing a standalone object, but really it's constructing the sub-object of a in place. That's wild. I'm curious if there are cases in which this abstraction leaks or fails altogether. If the constructor calls a virtual method, it will use the vtable, but that's consistent with the normal C++ rule for vtables of sub-objects during construction. If has a virtual base, the optimization can't work in general because the sub-object might have a different layout from the standalone object; there's some further discussion...

  Read more

  JL

  GL March 8, 2025

  In LLVM bug 34516, according to Richard Smith, this is a bug in standard wording and "guaranteed copy elision" cannot be applied to base subobjects because of layout difference. So your concern of virtual base is very valid.

  I think (1) either the current standard truly wants to require guaranteed copy elision for base subobjects, in which case any function returning a class object with a virtual base must know whether it's returning a most derived object or a base subobject (e.g., with a hidden flag or smuggling some information into the target storage pointer), (2) or guaranteed copy elision is...

  Read more

  Matt McCutchen March 8, 2025 · Edited

  FWIW to speculate about this here:

  any function returning a class object with a virtual base must know whether it’s returning a most derived object or a base subobject (e.g., with a hidden flag or smuggling some information into the target storage pointer)

  More concretely, the information that needs in order to construct a subobject is the VTT to pass to the subobject constructor. could take an extra parameter that gives the VTT or null to use the standalone constructor. (That has to be a separate case because the standalone constructor constructs the virtual bases,...

  Read more

- IK

  Ivan Kljajic March 7, 2025

  Unrealted maybe, but would letting a little bit of proposed syntactic sugar, like a pretty cast to void around the ternary, be enough to prevent a copy from being generated? That way one could just say “x() if cond else y()” without a discard being generated.

- 恩嘉

  許恩嘉

  So why doesn’t the standard allow copy elision for the ternary operator?

  One possibility that comes to my mind is that the result of the ternary operator may be used as an lvalue.

  ```
  (a==b ? c : d) = 42;
  ```

Stay informed

Get notified when new posts are published.