How to return smart pointers (shared_ptr), by reference or by value?

Asked 12 years, 7 months ago Modified 7 years, 5 months ago

Viewed 127k times

130

Let's say I have a class with a method that returns a `shared_ptr`.

What are the possible benefits and drawbacks of returning it by reference or by value?

Two possible clues:

- **Early object destruction.** If I return the `shared_ptr` by (const) reference, the reference counter is not incremented, so I incur the risk of having the object deleted when it goes out of scope in another context (e.g. another thread). Is this correct? What if the environment is single-threaded, can this situation happen as well?
- **Cost.** Pass-by-value is certainly not free. Is it worth avoiding it whenever possible?

Thanks everybody.

c++ return smart-pointers

Share

Improve this question

Follow

asked May 17, 2012 at 21:09

Vincenzo Pii

19.8k ● 9 ● 41 ● 50

2 Answers

Sorted by: Highest score (default)

146

Return smart pointers by value.

As you've said, if you return it by reference, you won't properly increment the reference count, which opens up the risk of deleting something at the improper time. That alone should be enough reason to not return by reference. Interfaces should be robust.

The cost concern is nowadays moot thanks to return value optimization (RVO), so you won't incur a increment-increment-decrement sequence or something like that in modern compilers. So the best way to return a `shared_ptr` is to simply return by value:

```
shared_ptr<T> Foo()
{
    return shared_ptr<T>(/* acquire something */);
};
```

This is a dead-obvious RVO opportunity for modern C++ compilers. I know for a fact that Visual C++ compilers implement RVO even when all optimizations are turned off. And with C++11's move semantics, this concern is even less relevant. (But the only way to be sure is to profile and experiment.)

If you're still not convinced, Dave Abrahams has an article that makes an argument for returning by value. I reproduce a snippet here; I highly recommend that you go read the entire article:

> Be honest: how does the following code make you feel?
>
> ```
> std::vector<std::string> get_names();
> ...
> std::vector<std::string> const names = get_names()
> ```
>
> Frankly, even though I should know better, it makes me nervous. In principle, when `get_names()` returns, we have to copy a `vector` of `string`s. Then, we need to copy it again when we initialize `names`, and we need to destroy the first copy. If there are N `string`s in the vector, each copy could require as many as N+1 memory allocations and a whole slew of cache-unfriendly data accesses > as the string contents are copied.
>
> Rather than confront that sort of anxiety, I've often fallen back on pass-by-reference to avoid needless copies:
>
> ```
> get_names(std::vector<std::string>& out_param );
> ...
> ```

> ```
> std::vector<std::string> names;
> get_names( names );
> ```
>
> Unfortunately, this approach is far from ideal.
>
> - The code grew by 150%
> - We've had to drop `const`-ness because we're mutating names.
> - As functional programmers like to remind us, mutation makes code more complex to reason about by undermining referential transparency and equational reasoning.
> - We no longer have strict value semantics for names.
>
> **But is it really necessary to mess up our code in this way to gain efficiency? Fortunately, the answer turns out to be no (and especially not if you are using C++0x).**

Share Improve this answer Follow

edited Jun 30, 2017 at 16:06
Ray Hulha
11.2k ● 5 ● 55 ● 56

answered May 17, 2012 at 21:13
In silico
52k ● 10 ● 152 ● 144

I don't know that I would say RVO makes the question moot since returning by reference decidedly makes RVO impossible. – Edward Strange May 17, 2012 at 23:18

@CrazyEddie: True, that's one of the reasons why I recommend that the OP return by value. – In silico May 18, 2012 at 2:41

Does the RVO rule, allowed by the standard, trump the rules about synchronization/happens-before relationships, guaranteed by the standard? – edA-qa mort-ora-y May 18, 2012 at 8:01

1 @edA-qa mort-ora-y: RVO is explicitly allowed even if it has side-effects. For example, if you have a `cout << "Hello World!";` statement in a default and copy constructor, you won't see two `Hello World!`s when RVO is in effect. However, this should not be a problem for properly-designed smart pointers, even w.r.t. synchronization. – In silico May 18, 2012 at 18:39

What if `shared_ptr` is a parameter, pass by value or const-reference? – Alfred Jan 28, 2021 at 12:21

26

Regarding *any* smart pointer (not just shared_ptr), I don't think it's ever acceptable to return a reference to one, and I would be very hesitant to pass them around by reference or raw pointer. Why? Because you cannot be certain that it will not be shallow-copied via a reference later. Your first point defines the reason why this should be a concern. This can happen even in a single-threaded environment. You don't need concurrent access to data to put bad copy semantics in your programs. You don't really control what your users do with the pointer once you pass it off, so don't encourage misuse giving your API users enough rope to hang themselves.

Secondly, look at your smart pointer's implementation, if possible. Construction and destruction should be darn close to negligible. If this overhead isn't acceptable, then don't use a smart pointer! But beyond this, you will also need to examine the concurrency architecture that you've got, because mutually exclusive access to the mechanism that tracks the uses of the pointer is going to slow you down more than mere construction of the shared_ptr object.

Edit, 3 years later: with the advent of the more modern features in C++, I would tweak my answer to be more accepting of cases when you've simply written a lambda that never lives outside of the calling function's scope, and isn't copied somewhere else. Here, if you wanted to save the very minimal overhead of copying a shared pointer, it would be fair and safe. Why? Because you can guarantee that the reference will never be mis-used.

Share Improve this answer Follow

edited Oct 24, 2015 at 20:53

answered May 17, 2012 at 21:17

user124493