Is returning by rvalue reference more efficient?

Asked 13 years ago Modified 4 years, 3 months ago Viewed 87k times

156

85

for example:

```
Beta_ab&&
Beta::toAB() const {
    return move(Beta_ab(1, 1));
}
```

c++ c++11 rvalue-reference

Share Follow

edited Oct 3, 2011 at 12:59
Johannes Schaub - litb
483k 124 873 1196

asked Jul 12, 2009 at 18:44
Neil G
30.7k 35 150 248

2 Answers

Sorted by:
Trending sort available
Highest score (default)

292

```
Beta_ab&&
Beta::toAB() const {
    return move(Beta_ab(1, 1));
}
```

This returns a dangling reference, just like with the lvalue reference case. After the function returns, the temporary object will get destructed. You should return `Beta_ab` by value, like the following

```
Beta_ab
Beta::toAB() const {
    return Beta_ab(1, 1);
}
```

Now, it's properly moving a temporary `Beta_ab` object into the return value of the function. If the compiler can, it will avoid the move altogether, by using RVO (return value optimization). Now, you can do the following

Your case of returning an rvalue reference would be a good idea in other occasions. Imagine you have a `getAB()` function which you often invoke on a temporary. It's not optimal to make it return a const lvalue reference for rvalue temporaries. You may implement it like this

```
struct Beta {
  Beta_ab ab;
  Beta_ab const& getAB() const& { return ab; }
  Beta_ab && getAB() && { return move(ab); }
};
```

Note that `move` in this case is not optional, because `ab` is neither a local automatic nor a temporary rvalue. Now, the *ref-qualifier* `&&` says that the second function is invoked on rvalue temporaries, making the following move, instead of copy

```
Beta_ab ab = Beta().getAB();
```

Share Follow edited Jul 12, 2009 at 20:28 answered Jul 12, 2009 at 19:54

Johannes Schaub - litb
483k 124 873 1196

64 I had always assumed the dangling reference problem went away automagically when the return type was an r-value reference. Glad I got that straighted out before it bit me. Stack smashing bugs suck. – deft_code Jul 15, 2009 at 3:03

33 :) Really, rvalue references are "just references" like lvalue references are. they don't copy or store anything. – Johannes Schaub - litb Mar 10, 2010 at 2:17

11 what does the const & qualifier on a member function more than a simple const ? – galinette Apr 11, 2014 at 11:33

4 +1-ed, but broken link: BoostCon09 Rvalue References 101 – Siu Ching Pong -Asuka Kenji- Mar 30, 2015 at 0:45

3 @galinette These are ref-qualifiers. – Malcolm Oct 20, 2015 at 10:55

0

It **can** be more efficient, for example, in a bit different context:

```
template <typename T>
T&& min_(T&& a, T &&b) {
    return std::move(a < b? a: b);
}
```

Your privacy

By clicking "Accept all cookies", you agree Stack Exchange can store cookies on your device and disclose information in accordance with our Cookie Policy.

Accept all cookies Customize settings

518 instructions for code above versus 481 for regular `std::min`.

Share Follow

answered Apr 9, 2018 at 17:15
wonder.mice
6,786 2 34 38

I'm confused by your answer. Had tried a similar (maybe) version but failed: ideone.com/4GyUbZ Could you explain why? – Deqing Apr 11, 2018 at 0:25

You used reference on temporary object in `for(:)` and integrating over deallocated object. Fix: ideone.com/tQVOal – wonder.mice Apr 11, 2018 at 17:42

11 isn't this answer actually wrong? for template parameter T, T&& is *not* an r-value reference, but rather a universal reference, and for that case, we should always call std::forward<T>, *not* std::move! Not to mention, that this answer directly contradicts the top-voted one above. – xdavidliu Nov 3, 2019 at 17:49

2 @xdavidliu this answer is a contrived example how returning by rvalue can be more efficient. `std::move()` is just used as an explicit cast to illustrate the point more clearly. It's not a code you would copy-paste into your project. It doesn't contradict top-voted answer, because there temporary object is created inside the function. Here returned object is one of the arguments (temporary objects are destroyed as the last step in evaluating the full-expression that (lexically) contains the point where they were created). – wonder.mice Apr 15, 2020 at 19:08

4 @wonder.mice then please replace T with std::string; there's no need to use templates at all here, and using T&& as a r-value reference is just awful style that unnecessarily confuses people new to templates and r-values. – xdavidliu Apr 15, 2020 at 19:14