rvalue reference or forwarding reference?

Asked 4 years, 7 months ago Modified 1 year, 4 months ago Viewed 877 times

5

I know that for the following function

```
template <typename T>
void do_something(T&& arg);
```

the function parameter is a forwarding reference. But in the following case is it still a forwarding reference or an rvalue reference?

```
template <typename T>
class MyClass
{
    void do_something(T&& arg);
};
```

I think it is still a forwarding reference, but I'm not sure. Furthermore, I'd like to know what can be done to enforce an rvalue reference or a forwarding reference, if the result is not what I intended.

c++

Share Edit Follow Flag

edited Nov 22, 2020 at 16:10 Enlico 17.2k ● 5 ● 36 ● 74

asked Sep 14, 2017 at 8:08 Martin Fehrs 751 ● 4 ● 12

2 Answers

Sorted by: Highest score (default)

10

It's an rvalue reference. Forwarding references can only appear in a deduced context. This is just a member function that accepts an rvalue reference to the **class** template parameter.

You can't force a forwarding reference to be an rvalue reference if you want to maintain template argument deduction for functions. If you don't mind specifying the template argument all over the place, then this will always and only ever give an rvalue reference:

```
template<typename T> struct identity { using type = T; };

template<typename T> void func(typename identity<T>::type&&);
```

In retrospect, there actually is a way to maintain deduction but force only rvalue refs to be accepted (besides the self documenting one in **Simple**'s answer). You can provide a deleted lvalue overload:

```
template<typename T>
void func(T&) = delete;

template<typename T>
void func(T&& s)
{
    // ...
}
```

The lvalue overload is more specialized when passed an lvalue. And on account of being deleted, will give a somewhat clear error message.

Share Edit Follow Flag

edited Nov 22, 2020 at 16:10 Enlico 17.2k ● 5 ● 36 ● 74

answered Sep 14, 2017 at 8:09 StoryTeller - Unslander Monica 159k ● 21 ● 357 ● 433

1 Does decuced context mean that the type is automatically deduced from the passed argument? If I understand you correctly, T is not deduced, because it's part of the class signature. – Martin Fehrs Sep 14, 2017 at 8:15

@MartinKalbfuß - Precisely. A deduced context is when you call the free function version of do_something. – StoryTeller - Unslander Monica Sep 14, 2017 at 8:16

5

> Furthermore I like to know, what can be done to enforce an rvalue reference

If you always want an rvalue reference in a deduced context (and not a forwarding reference), then you can use this:

```
template<
    typename T,
    typename = std::enable_if_t<!std::is_lvalue_reference<T>::value>
>
using rval_ref = T&&;

template<typename T>
void foo(rval_ref<T> s)
{
    // ...
}
```

`foo` can only be called with an rvalue, and `T` will not be a reference (i.e. if you call `foo` with `std::string&&`, then `T` will be `std::string`).

Share Edit Follow Flag

answered Sep 14, 2017 at 8:30 Simple 13.3k ● 2 ● 40 ● 46

Not bad. Got rid of the dependent type and used SFINAE to block lvalue refs. +1 – StoryTeller - Unslander Monica Sep 14, 2017 at 8:35