Futures vs. Promises

Asked 8 years, 8 months ago Active 5 years, 4 months ago Viewed 55k times

142

63

I'm confusing myself with difference between a future and a promise.

Obviously, they have different methods and stuff, but what is the actual use case?

Is it?:

- when I'm managing some async task, I use future to get the value "in future"
- when I'm the async task, I use promise as the return type to allow the user get a future from my promise

c++ c++11 promise future

Share Follow

edited Jul 25 '14 at 3:42
Bergi
513k 108 820 1163

asked Sep 27 '12 at 11:20
Šimon Tóth
33.4k 18 94 135

1 I wrote a bit about this in this answer. – Kerrek SB Sep 27 '12 at 11:33

1 possible duplicate of What is std::promise? – Nicol Bolas Sep 27 '12 at 14:28

1 Answer

Active Oldest Votes

173

Future and Promise are the two separate sides of an asynchronous operation.

`std::promise` is used by the "producer/writer" of the asynchronous operation.

`std::future` is used by the "consumer/reader" of the asynchronous operation.

The reason it is separated into these two separate "interfaces" is to **hide** the "write/set" functionality from the "consumer/reader".

```
auto promise = std::promise<std::string>();

auto producer = std::thread([&]
{
    promise.set_value("Hello World");
});

auto future = promise.get_future();
```

```
});

producer.join();
consumer.join();
```

One (incomplete) way to implement std::async using std::promise could be:

```
template<typename F>
auto async(F&& func) -> std::future<decltype(func())>
{
    typedef decltype(func()) result_type;

    auto promise = std::promise<result_type>();
    auto future  = promise.get_future();

    std::thread(std::bind([=](std::promise<result_type>& promise)
    {
        try
        {
            promise.set_value(func()); // Note: Will not work with
std::promise<void>. Needs some meta-template programming which is out of scope
for this question.
        }
        catch(...)
        {
            promise.set_exception(std::current_exception());
        }
    }, std::move(promise))).detach();

    return std::move(future);
}
```

Using `std::packaged_task` which is a helper (i.e. it basically does what we were doing above) around `std::promise` you could do the following which is more complete and possibly faster:

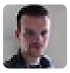
```
template<typename F>
auto async(F&& func) -> std::future<decltype(func())>
{
    auto task   = std::packaged_task<decltype(func())()>(std::forward<F>
(func));
    auto future = task.get_future();

    std::thread(std::move(task)).detach();

    return std::move(future);
}
```

Note that this is slightly different from `std::async` where the returned `std::future` will when destructed actually block until the thread is finished.

Share Follow

edited Jan 27 '16 at 9:03

answered Sep 27 '12 at 11:24



ronag
43.5k 23 112 204

4 @taras suggests that returning `std::move(something)` is useless and it also hurts (N)RVO.

8 For those who are still confused, see this answer. – kawing-chiu Aug 11 '16 at 1:39

3 That is a one time producer - consumer, IMHO that is not really a producer - consumer pattern. – Martin Meeser Aug 12 '16 at 16:55