

진리는어디에/C++20

[C++20] 코루틴(Coroutine) - 활용

찾기 2021. 4. 3. 18:45



코루틴.h

```
1  #ifndef _COROUTINE_H_
2  #define _COROUTINE_H_
3
4  #include <coroutine>
5  #include <memory>
6
7  template <class T, class INITIAL_SUSPEND = std::suspend_always>
8  class Coroutine
9  {
10 private :
11     class Impl;
12
13     struct promise_base
14     {
15         INITIAL_SUSPEND initial_suspend()
16         {
17             return INITIAL_SUSPEND{};
18         }
19
20         std::suspend_always final_suspend() noexcept
21         {
22             return {};
23         }
24
25         void unhandled_exception()
26         {
27             throw std::exception("unhandled exception");
28         }
29     };
30
31     };
32
33     template <class R>
34     struct promise_type_impl :
35     public promise_base
36     {
37     };
38 }
```

58%

22%

5%

19%

```

42
43     std::suspend_always yield_value(R&& value)
44     {
45         this->value = value;
46         return {};
47     }
48
49     std::suspend_always yield_value(const R& value)
50     {
51         this->value = value;
52         return {};
53     }
54
55     void return_void()
56     {
57         impl->done = true;
58     }
59 };
60
61 template <>
62 struct promise_type_impl<void> : public promise_base
63 {
64     Coroutine get_return_object()
65     {
66         return Coroutine{ std::make_shared<Impl>
67             (std::coroutine_handle<promise_type_impl>::from_promise(*this)) };
68     }
69
70     void return_void()
71     {
72     };
73
74 public :
75     typedef promise_type_impl<typename T> promise_type;
76
77 public :
78     Coroutine()
79     : impl(nullptr)
80     {
81     }
82
83     Coroutine(std::shared_ptr<Impl> impl)
84     : impl(impl)
85     {
86     }
87
88     Coroutine(const Coroutine& other)
89     : impl(other.impl)
90     {
91     }
92
93     bool operator()()
94     {
95         return resume();
96     }
97
98     bool resume()
99     {
100         if (true == done())
101         {
102             return false;
103         }
104
105         impl->handle.resume();
106
107         if (impl->done)
108         {
109             return false;
110         }
111         return true;
112     }
113
114     promise_type& promise()
115     {
116         return impl->handle.promise();
117     }
118

```



58%



22%

5%

19%



```

126         return !impl->handle || impl->handle.done();
127     }
128
129     Coroutine& operator = (const Coroutine& other)
130     {
131         impl = other.impl;
132     }
133
134     struct iterator
135     {
136         explicit iterator(Coroutine* coroutine)
137             : coroutine(coroutine)
138         {
139         }
140
141         const T& operator* () const
142         {
143             return coroutine->promise().value;
144         }
145
146         iterator& operator++()
147         {
148             coroutine->resume();
149             return *this;
150         }
151
152         iterator& operator++(int)
153         {
154             coroutine->resume();
155             return *this;
156         }
157
158         bool operator == (std::default_sentinel_t) const
159         {
160             return coroutine->done();
161         }
162     private:
163         Coroutine* coroutine;
164     };
165
166     iterator begin()
167     {
168         if (nullptr == impl)
169         {
170             return iterator{ nullptr };
171         }
172
173         if(impl->handle)
174         {
175             impl->handle.resume();
176         }
177
178         return iterator{ this };
179     }
180
181     std::default_sentinel_t end()
182     {
183         return {};
184     }
185     private:
186     class Impl
187     {
188     public:
189         Impl(std::coroutine_handle<promise_type> handle)
190             : handle(handle)
191             , done(false)
192         {
193         }
194
195         ~Impl()
196         {
197             if (true == (bool)handle)
198             {
199                 handle.destroy();
200             }
201         }
202
203         std::coroutine_handle<promise_type> handle;

```



58%



22%



5%

19%

예

```

1 // main.cpp
2 #include <iostream>
3 #include "Coroutine.h"
4
5 Coroutine<void> lazily_start()
6 {
7     std::cout << "\tlazily_start 1" << std::endl;
8     co_await std::suspend_always{};
9     std::cout << "\tlazily_start 2" << std::endl;
10 }
11
12 Coroutine<void, std::suspend_never> eagerly_start()
13 {
14     std::cout << "\teagerly_start 1" << std::endl;
15     co_await std::suspend_always{};
16     std::cout << "\teagerly_start 2" << std::endl;
17 }
18
19 Coroutine<int> yield(int start, int end)
20 {
21     for (int i = start; i < end; i++)
22     {
23         co_yield i;
24     }
25 }
26
27 int main()
28 {
29     {
30         std::cout << "==== lazily_start example ===" << std::endl;
31         Coroutine<void> coroutine = lazily_start();
32         std::cout << "main 1" << std::endl;
33         coroutine();
34         std::cout << "main 2" << std::endl;
35         coroutine();
36         /* OUTPUT
37         ==== lazily_start example ====
38         main 1
39             lazily_start 1
40         main 2
41             lazily_start 2
42         */
43     }
44
45     {
46         std::cout << "==== eagerly_start example ===" << std::endl;
47         Coroutine<void, std::suspend_never> coroutine = eagerly_start();
48         std::cout << "main 1" << std::endl;
49         coroutine();
50         std::cout << "main 2" << std::endl;
51         coroutine();
52         /* OUTPUT
53         ==== eagerly_start example ====
54         eagerly_start 1
55         main 1
56             eagerly_start 2
57         main 2
58         */
59     }
60
61     {
62         std::cout << "==== yield while loop example ===" << std::endl;
63         Coroutine<int> coroutine = yield(0, 5);
64
65         int i = 0;
66
67         while (true == coroutine.resume())
68         {

```

58%

22%

5%

19%

```

76     main 1
77         yield value 1
78     main 2
79         yield value 2
80     main 3
81         yield value 3
82     main 4
83         yield value 4
84     */
85 }
86
87 {
88     std::cout << "==== yield ranged-for example ====" << std::endl;
89     Coroutine<int> coroutine = yield(5, 10);
90
91     int i = 0;
92
93     for (int value : coroutine)
94     {
95         std::cout << "main " << i++ << std::endl;
96         std::cout << "\tyield value " << value << std::endl;
97     }
98     /* OUTPUT
99     ==== yield ranged-for example ====
100    main 0
101        yield value 5
102    main 1
103        yield value 6
104    main 2
105        yield value 7
106    main 3
107        yield value 8
108    main 4
109        yield value 9
110    */
111 }
112 }

```

결과

```

==== lazily_start example ====
main 1
    lazily_start 1
main 2
    lazily_start 2
==== eagerly_start example ====
eagerly_start 1
main 1
    eagerly_start 2
main 2
==== yield while loop example ====
main 0
    yield value 0
main 1
    yield value 1
main 2
    yield value 2
main 3
    yield value 3
main 4
    yield value 4
==== yield ranged-for example ====
main 0
    yield value 5
main 1
    yield value 6
main 2
    yield value 7
main 3
    yield value 8
main 4
    yield value 9

```

○
○
○



58%



22%



5%

19%

Shutterstock 무료 평가판
Shutterstock

공감

구독하기

태그 #C++20, #코루틴

'진리논어디에/C++20' Related Articles

C++20

2020

Core Language

• constexpr comparison operator
• constexpr literals as template arguments
• constexpr virtual functions
• Definition of module namespace
• constexpr initializers
• constexpr lambda improvements
• constexpr standard attributes
• constexpr namespace and constexpr keyword
• constexpr_location

Library

• Calendar and time-zone
• std::lspan as a view on a contiguous array
• constexpr containers such as std::array and std::vector
• std::format

• std::atomic
• std::atomic_wait
• std::atomic_waiter
• std::atomic_waiter_waiter
• std::atomic_waiter_waiter_waiter

NO IMAGE

NO IMAGE

NO IMAGE

[C++20] 개념

[C++20] 범위

[C++20] 코루틴(Coroutine) - done()

[C++20] 코루틴(Coroutine) - co_yield



유익한 글이었다면 공감(♥) 버튼 꼭!! 추가 문의 사항은 댓글로!!

이름

암호

☐ 비밀

여러분이 사랑한 글이 있다면 댓글로 남겨주세요! 댓글 달고 갈꺼지?

58%

22%

5%

19%

댓글달기



✓

	58%	22%	5%	19%
▲				
▼				