진리는어디에/C++20

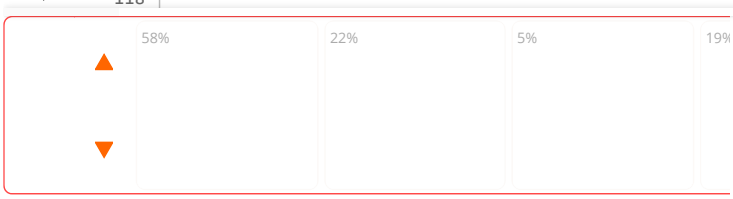
[C++20] 코루틴(Coroutine) - 활용

찾기 2021. 4. 3. 18:45

코루틴.h

```
#ifndef _COROUTINE_H_
#define _COROUTINE_H_

#include <coroutine>
#include <memory>

template <class T, class INITIAL_SUSPEND = std::suspend_always>
class Coroutine
{
private :
    class Impl;

    struct promise_base
    {
        INITIAL_SUSPEND initial_suspend()
        {
            return INITIAL_SUSPEND{};
        }

        std::suspend_always final_suspend() noexcept
        {
            return {};
        }

        void unhandled_exception()
        {
            throw std::exception("unhandled exception");
        }

    };

    template <class R>
    struct promise_type_impl :
public promise_base
    {
```

```
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>
                (std::coroutine_handle<promise_type_impl>::from_promise(*this)) };
67          }
68
69          void return_void()
70          {
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
```



```
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%

```
#endif
```

예

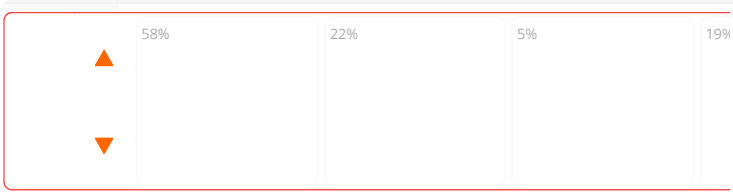
```
// main.cpp
#include <iostream>
#include "Coroutine.h"

Coroutine<void> lazily_start()
{
    std::cout << "\tlazily_start 1" << std::endl;
    co_await std::suspend_always{};
    std::cout << "\tlazily_start 2" << std::endl;
}

Coroutine<void, std::suspend_never> eagerly_start()
{
    std::cout << "\teagerly_start 1" << std::endl;
    co_await std::suspend_always{};
    std::cout << "\teagerly_start 2" << std::endl;
}

Coroutine<int> yield(int start, int end)
{
    for (int i = start; i < end; i++)
    {
        co_yield i;
    }
}

int main()
{
    {
        std::cout << "==== lazily_start example ====" << std::endl;
        Coroutine<void> coroutine = lazily_start();
        std::cout << "main 1" << std::endl;
        coroutine();
        std::cout << "main 2" << std::endl;
        coroutine();
        /* OUTPUT
        ==== lazily_start example ====
        main 1
                lazily_start 1
        main 2
                lazily_start 2
        */
    }

    {
        std::cout << "==== eagerly_start example ====" << std::endl;
        Coroutine<void, std::suspend_never> coroutine = eagerly_start();
        std::cout << "main 1" << std::endl;
        coroutine();
        std::cout << "main 2" << std::endl;
        coroutine();
        /* OUTPUT
        ==== eagerly_start example ====
                eagerly_start 1
        main 1
                eagerly_start 2
        main 2
        */
    }

    {
        std::cout << "==== yield while loop example ====" << std::endl;
        Coroutine<int> coroutine = yield(0, 5);

        int i = 0;

        while (true == coroutine.resume())
        {
```

58% 22% 5% 19%

```
        main 1
                yield value 1
        main 2
                yield value 2
        main 3
                yield value 3
        main 4
                yield value 4
        */
    }

    {
        std::cout << "==== yield ranged-for example ====" << std::endl;
        Coroutine<int> coroutine = yield(5, 10);

        int i = 0;

        for (int value : coroutine)
        {
            std::cout << "main " << i++ << std::endl;
            std::cout << "\tyield value " << value << std::endl;
        }
        /* OUTPUT
        ==== 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
        */
    }
}
```

결과

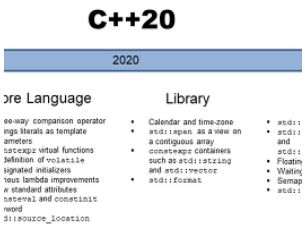
```
==== lazily_start example ====
main 1
        lazily_start 1
main 2
        lazily_start 2
==== eagerly_start example ====
        eagerly_start 1
main 1
        eagerly_start 2
main 2
==== yield while loop example ====
main 0
        yield value 0
main 1
        yield value 1
main 2
        yield 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
```



공감 구독하기

태그 #C++20, #코루틴

'진리는어디에/C++20' Related Articles



[C++20] 개념

NO IMAGE

[C++20] 범위

NO IMAGE

[C++20] 코루틴(Coroutine) - done()

NO IMAGE

[C++20] 코루틴(Coroutine) - co_yield

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

이름 암호 비밀

댓글달기

1 ··· 113 114 115 116 117 118 119 120 121 ··· 342