第33章 协程

《现代C++语言核心特性解析》 谢丙堃

协程的使用方法

- 协程是一种可以被挂起和恢复的函数，它提供了一种创建异步代码的方法。

- 基于MSVC的/await编译选项进行讲解
  - 扩展了标准库，提供了一些辅助类

协程的使用方法

- co_await

```
std::future<int> foo() {
    std::cout << "call foo\n";
    std::this_thread::sleep_for(3s);
    co_return 5;
}
std::future<std::future<int>> bar() {
    std::cout << "call bar\n";
    std::cout << "before foo\n";
    auto n = co_await std::async(foo);  // 挂起点
    std::cout << "after foo\n";
    co_return n;
}
int main() {
    std::cout << "before bar\n";
    auto i = bar();
    std::cout << "after bar\n";
    i.wait();
    std::cout << "result = " << i.get().get();
}
```

输出:

```
before bar
call bar
before foo
after bar
call foo
after foo
result = 5
```

协程的使用方法

- co_return

```
set_return_future_value(5);
set_future_ready();
```

协程的使用方法

- co_yield

```
std::experimental::generator<int> foo() {
    std::cout << "begin" << std::endl;
    for (int i = 0; i < 10; i++) {
        co_yield i;
    }
    std::cout << "end" << std::endl;
}
int main() {
    for (auto i : foo()) {
        std::cout << i << std::endl;
    }
}
```

输出:

```
begin
0
1
2
3
4
5
6
7
8
9
end
```

co_await运算符原理

- 不是所有对象都能等待

```
co_await std::string{ "hello" };

error C3312: no callable 'await_resume' function found for type 'std::string'
error C3312: no callable 'await_ready' function found for type 'std::string'
error C3312: no callable 'await_suspend' function found for type 'std::string'
```

- 规则说明
  - await_ready函数用于判定可等待体是否已经准备好
  - await_suspend函数调度协程的执行流程
  - await_resume函数一个接收异步执行结果

co_await运算符原理

- 同步的情况

```
class awaitable_string : public std::string {
public:
    using std::string::string;
    bool await_ready() const { return true; }
    void await_suspend(std::experimental::coroutine_handle<> h) const {}
    std::string await_resume() const { return *this; }
};
std::future<std::string> foo() {
    auto str = co_await awaitable_string{ "hello" };
    co_return str;
}
int main() {
    auto s = foo();
    std::cout << s.get();
}
```

co_await运算符原理

- 异步的情况

```
class awaitable_string : public std::string {
public:
    using std::string::string;
    bool await_ready() const { return false; }
    void await_suspend(std::experimental::coroutine_handle<> h) const {
        std::thread t{ [h] {
            // 模拟复杂操作，用时3秒
            std::this_thread::sleep_for(3s);
            h(); }
        };
        t.detach();
    }
    std::string await_resume() const { return *this; }
};
```

co_await运算符原理

- co_await运算符的重载

```
awaitable_string operator co_await(std::string&& str)
{
    return awaitable_string{ str };
}

std::future<std::string> foo()
{
    auto str = co_await std::string{ "hello" };
    co_return str;
}
```

co_yield运算符原理

- 缺少promise_type

```
struct my_int_generator {};

my_int_generator foo() {
    for (int i = 0; i < 10; i++) {
        co_yield i;
    }
}
```

```
error C2039: 'promise_type': is not a member of
'std::experimental::coroutine_traits<my_int_generator>'
```

co_yield运算符原理

- 定义promise_type

```
struct promise_type {
    int* value_ = nullptr;
    my_int_generator get_return_object() {
        return my_int_generator{ *this };
    }
    auto initial_suspend() const noexcept {
        return suspend_always{};
    }
    auto final_suspend() const noexcept {
        return suspend_always{};
    }
    auto yield_value(int& value) {
        value_ = &value;
        return suspend_always{};
    }
    void return_void() {}
};
```

co_yield运算符原理

- 重新定义my_int_generator

```
struct my_int_generator {
    struct promise_type {...};

    explicit my_int_generator(promise_type& p)
        : handle_(coroutine_handle<promise_type>::from_promise(p)) {}
    ~my_int_generator() {
        if (handle_) {
            handle_.destroy();
        }
    }
    coroutine_handle<promise_type> handle_;
};
```

co_yield运算符原理

- 重新定义my_int_generator

```
struct my_int_generator {
    int next() {
        if (!handle_ || handle_.done()) {
            return -1;
        }
        handle_();
        return handle_.promise().value_;
    }
};

int main()
{
    auto obj = foo();
    std::cout << obj.next() << std::endl;
    std::cout << obj.next() << std::endl;
    std::cout << obj.next() << std::endl;
}
```

co_return运算符原理

- promise_type

```
struct promise_type {
    int value_ = 0;
    my_int_return get_return_object() {
        return my_int_return{ *this };
    }

    auto initial_suspend() const noexcept {
        return suspend_never{};
    }
    auto final_suspend() const noexcept {
        return suspend_always{};
    }

    void return_value(int value) {
        value_ = value;
    }
};
```

promise_type的其他功能

- await_transform

```
struct promise_type {
    ...
    awaitable await_transform(expr e) {
        return awaitable(e);
    }
};

co_await expr;

// 最终会转换为:

co_await promise.await_transform(expr);
```

感谢您的观看

欢迎关注