OTUS

Современный C++

Обзор C++20.

otus.ru

REC

Меня хорошо видно && слышно?

Ставим “+”, если все хорошо
“-”, если есть проблемы

Тема вебинара

C++: Модули, корутины, концепты

Владимир Щерба

C++/Rust Backend, Quadrivium

Telegram: @harrm

План вебинара

- Концепты
- format
- spaceship operator
- ranges
- Модули
- Корутины

Концепты

Проблема

- В шаблонном коде под любой шаблонный параметр подходит ограниченное множество типов. Эту информацию нужно донести пользователю кода.

Решения

- **SFINAE** в чистом виде: компилятор выдаст полотно ошибок в котором где-то будет нужная информация.

  По сигнатуре функции / описанию класса непонятно ничего.
- **std::enable_if и т. п.**: код будет полотном замысловатых шаблонных конструкций и компилятор выдаст полотно ошибок в которых будет чуть более понятная информация.

  По сигнатуре функции / описанию класса можно понять, какие параметры ожидаются. Но с трудом.

Концепты

Решение C++-20

- По-человечески выглядящие понятно именованные описания ограничений шаблонных параметров.
- Два новых ключевых слова: **concept** и **requires**.
- **concept**: именованный набор **ограничений** шаблонного параметра.
- **Ограничения** шаблонного параметра: другой концепт, булево выражение, или логическая функция (И/ИЛИ) из ограничений.
- **requires**: очень удобный способ получить булево выражение из описания вещей которые мы хотим уметь делать с типом.

format

Проблема

Форматирование строк -- очень распространенная задача. Например, для логов.

Решения

- семейство printf - пока, безопасность и надежность.
- iostream - неудобно.

Решение C++-20

- Принять в стандартную библиотеку классную fmtlib.
- Apple Clang пока не поддерживает.

Spaceship operator

Проблема

Многим классам нужны операторы сравнения, но писать их вручную скучно.

Решение

Пусть компилятор генерирует их за нас.

Решение C++-20

Благодаря оператору ⇔ можно либо сказать компилятору сгенерировать все операторы сравнения за нас, либо одним методов их все задать самостоятельно.

Ranges

Проблема

Итераторы это мощная абстракция, но не всегда удобная.

Решение

Расширить STL более практичной абстракцией -- диапазонами (ranges).

Решение C++-20

Ranges позволяют писать более лаконичный и выразительный код для обработки последовательностей.

Модули

Проблема

C++-код надо как-то структурировать, как минимум чтобы писать библиотеки.

Решения

Единицы трансляции и препроцессор: компилятор обрабатывает все .cpp файлы независимо, все что нужно каждому .cpp файлу напрямую копируется в него через **#include**.

Модули

Проблемы со старой системой

- Просто копировать во все .cpp файлы отдельно нужный им код плохо для времени компиляции.
- Препроцессор ничего не знает о C++ и **#include** просто вбрасывает в файл все свое содержимое, включая **макросы**. Макросы могут сделать с кодом что угодно.
- Трудно писать инструменты разработки, так как им трудно будет разобраться что откуда взялось.

Модули

Решение C++-20

- Код разбивается на единицы модулей - новый вид единиц трансляции.
- Множество единиц модулей объединяется в модуль: программист явно указывает, что из модуля видно другим модулям, и никакой код копипастить не нужно.
- Макросы экспортировать нельзя. И хорошо.
- Модули не заменяют пространства имен.
- Компилятор теперь обрабатывает единицы трансляции не независимо, а в порядке в котором они друг друга импортируют.

Модули

- *Module interface unit* - `export module` - единица, которая экспортирует имена из модуля.
- *Module implementation unit* - единица, которая ничего не экспортирует.
- *Primary module interface unit* - уникальная единица, которая экспортирует имя модуля.
- *Module partition interface unit* - единица, которая экспортирует раздел модуля.
- *Module partition implementation unit* - единица, которая реализует раздел модуля.

Модули

- Компиляторы все еще поддерживают модули постольку-поскольку.
- Реализации между разными компиляторами и на разных платформах несовместимы.
- Реальных проектов использующих модули я не видел.
- CMake поддерживает их со вчерашнего дня: https://gitlab.kitware.com/cmake/cmake/-/issues/18355

Корутины (сопрограммы)

Проблема

Асинхронный код на C++ писать неудобно, так как функции по своей природе синхронные.

Решения

- Писать неудобно через колбеки.
- Использовать готовую большую сложную библиотеку которая реализует другой механизм или написать свою.

Решение C++-20

- Использовать большой и сложный синтаксис корутин чтобы компилятор делал магию.

Асинхронное программирование

Есть две функции: **network_request** и **compute_big_numbers**.

network_request отправляет запрос по сети и ждет ответ.

compute_big_numbers считает большое число, используя процессор.

Они занимают примерно одинаковое время.

- вызвать **network_request**
- ждем ответа, процессор простаивает
- вернуться из **network_request**
- вызвать **compute_big_numbers**
- считаем число, процессор нагружен
- вернуться из **compute_big_numbers**

Асинхронное программирование

Как хотелось бы:

- вызвать **network_request**
- ждем ответа, приостанавливаем (suspend) **network_request**
- вызвать **compute_big_numbers**
- считаем число, процессор нагружен
- вернуться из **compute_big_numbers**
- ответ пришел, возобновляем (resume) **network_request**
- **network_request** обрабатывает ответ и завершается окончательно

В итоге мы потратили в *два раза* меньше времени.

А как же потоки?

- Потоки довольно **тяжеловесная** штука так как требует **вмешательства ОС**.
- Если **network_request** и **compute_big_numbers** выполняются на разных ядрах, одно все равно будет простаивать.
- **Переключение** между потоками происходит принудительно и почти неконтролируемо.
- Данные, доступные из разных потоков, тяжело и дорого **синхронизовать** (т. е. защищать от **гонок данных**)

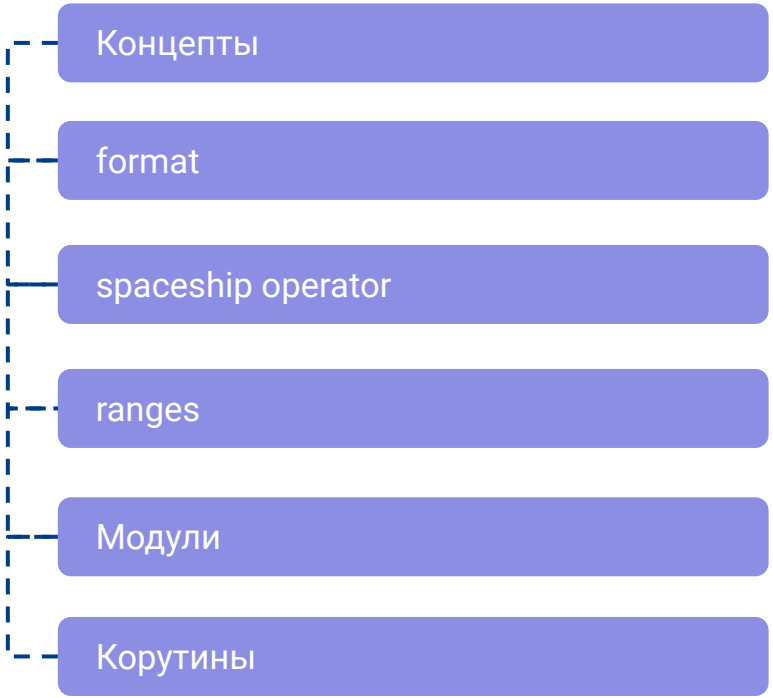
Конкурентность

- Выполнение нескольких операций "одновременно".
- Переключаемся между операциями в процессе выполнения.
- Два вида: кооперативная и принудительная.
- Корутины - механизм кооперативной конкурентности.
- Поэтому с ними легче избегать гонок данных и они в целом дешевле по производительности.

Корутины C++-20

- Новые ключевые слова: **co_await**, **co_return**, **co_yield**.
- Любая функция которая использует одно их них - корутина.
- Новые типы в библиотеке: **coroutine_handle**, **coroutine_traits**, **suspend_never**, **suspend_always** и несколько других.
- Новый вид функций - корутины.
- Кроме операций "вызов" и "возврат" появились операции "приостановка" и "возобновление" (suspend и resume).
- Два основных сценария использования - генераторы и асинхронные функции.
- Компилятор ничего не знает о поведении корутин и тут придется постараться.

Итоги вебинара



Материалы

- https://clang.llvm.org/docs/StandardCPlusPlusModules.html
- https://www.youtube.com/watch?v=HddFGPTAmtU
- https://www.youtube.com/watch?v=8C8NnE1Dg4A
- https://learn.microsoft.com/en-us/cpp/cpp/modules-cpp?view=msvc-170
- https://fmt.dev/latest/index.html
- https://en.cppreference.com/w/cpp/language/default_comparisons
- https://www.youtube.com/watch?v=5X803cXe02Y

Делимся впечатлениями

Q&A

Заполните, пожалуйста, опрос о занятии по ссылке в чате

Спасибо!