Проектирование больших систем на C++

Коноводов В. А.

кафедра математической кибернетики ВМК

Лекция 12
01.12.2017

Обработка ошибок

Основные способы обработки ошибок:

1. Возвращение ошибок
2. Обработка исключений

Возвращение ошибок

Будем возвращать из функции 1 (или `false`), если произошла ошибка.
Чем это плохо?

- Чтобы понять, в чем ошибка, нужно совершить дополнительные действия.
- Ошибки могут произойти в каждой строчке кода.
- Код обработки ошибок обычно плохо тестируется.
- Сложно обработать ошибки на большой глубине стека вызовов.

std::optional (C++17)

Шаблонный класс для хранения опционального значения.

```
template<class T>
class optional;
```

- `operator*` или метод `value` — доступ к хранимому значению.
- Метод `has_value` — проверка того, что есть значение.
- Метод `value_or` возвращает хранимое значение или переданный параметр, если значения нет.
- Пустое значение `std::nullopt`.

std::optional (C++17)

Удобно возвращать объекты этого класса в функциях, где возможна какая-то ошибка:

```
std::optional<int> GetCount(const TParam& param) {
    auto it = Params2Int.find(param);
    if (it != Params2Int.end()) {
        return it->second;
    } else {
        return {};
        // or:
        // return std::nullopt;
        // return std::optional<int>();
    }
}

int main() {
    // ...
    if (auto count = GetCount(param)) {
        std::cout << "res = " << *count << std::endl;
    }
}
```

Исключения: напоминание

Механизм исключений — наиболее предпочтительный механизм передачи сообщений об ошибке в C++.
Строки кода, в которых может сгенерироваться ошибка, заключаются в обертку

```
try {
    /*here may be error*/
} catch (/*what*/) {
    /*what to do*/
} catch (/*what*/) {
    /*what to do*/
} ...
```

Когда происходит исключение, генерируется некоторый объект исключения. Этот объект любой природы.
Среди всех `catch` будет выбрана лучшая функция, соответствующая нашему аргументу. Если текущий блок не может обработать исключение, то оно «проваливается» дальше на уровень выше.

Примеры исключений

- Исключение `std::bad_alloc` при нехватке динамической памяти.
- Исключение `std::length_error` при попытке сделать размер вектора или строки больше, чем `max_size()`.
- Исключение `std::out_of_range` при попытке функцией `at()` обратиться к контейнеру по некорректному индексу.

Исключения: catch

- `catch (MyException& ex) { ex.s = "error"; throw; }`

  У этого единственного объекта ex было изменено поле s, его кидаем дальше и он изменен.

- `catch (MyException& ex) { ex.s = "error"; throw ex; }`

  Генерируем новое исключение с объектом ex, вызывается конструктор копирования, а старый уничтожится.

- `catch (MyException ex) { ex.s = "error"; throw ex; }`

  Изменили копию, копию скопировали и бросили.

- `catch (MyException ex) { ex.s = "error"; throw; }`

  Изменили копию, а кинули то же исключение (исходный неизмененный объект).

Исключения и retry

```
for (int i = 1; ; ++i) {
    try {
        auto params = GetParams();
        auto result = GetResult(params);
        return;
    } catch (const std::exception& e) {
        if (i == MaxRetryCount) {
            std::cerr << "Error" << std::endl;
            throw;
        }
        std::cerr << "Error on " << i << std::endl;
    } catch (...) {
        std::cerr << "Fatal error" << std::endl;
        std::terminate();
    }
}
```

Исключения и retry

```
template <class TFunc>
auto Retry(TFunc func, int maxAttemptsCount) {
    for (int i = 1; ; ++i) {
        try {
            return f();
        } catch (const std::exception& e) {
            std::cerr << "Error on " << i << std::endl;
            if (i == maxAttemptsCount) {
                std::cerr
                    << "max attempts has been reached"
                    << std::endl;
                throw;
            }
        }
    }
}
```

Исключения стандартной библиотеки

Стандартная библиотека предоставляет иерархию классов исключений. Базовый класс `std::exception`.

- `logic_error`
  - `invalid_argument`
  - `domain_error`
  - `length_error`
  - `...`
- `runtime_error`
  - `overflow_error`
  - `range_error`
  - `...`
- `bad_weak_ptr`
- `bad_cast`
- `...`

std::exception_ptr (C++11)

- Хранит в себе исключения и имеет семантику указателя.
- Сохранить исключение можно через `std::current_exception`.
- Информация о типе исключения не теряется.
- Повторно сгенерировать исключение из этого объекта можно через `std::rethrow_exception`.

```
int main() {
    std::exception_ptr eptr;
    try {
        throw std::out_of_range("some info");
    } catch (...) {
        eptr = std::current_exception();
    }
    //...
    if (eptr) {
        std::rethrow_exception(eptr);
    }
}
```

Value Or Exception

`std::optional` не работает с исключениями. Создадим класс, объекты которого хранят в себе либо значение, либо `std::exception_ptr` сгенерированного исключения.

Как создать тип, объекты которого хранят значения одного из заданных типов?

- `union`
- `std::variant` (C++17)

std::variant (C++17)

Это такой union, который знает, какой именно тип он хранит.

```
std::variant<int, char, double> v;
v = 5;
std::cout << std::get<int>(v);

// std::cout << std::get<double>(v);
// исключение std::bad_variant_access

// std::cout << std::get<float>(v);
// не компилируется

auto p = std::get_if<char>(&v); // nullptr
```

Value Or Exception

```
template <typename T>
class TValueOrError {
    std::variant<T, std::exception_ptr> valueOrError;
  public:
    TValueOrError(std::exception_ptr eptr) : valueOrError(eptr) {}
    TValueOrError(T&& val) : valueOrError(std::move(val)) {}
    TValueOrError(const T& val) : valueOrError(val) {}

    bool IsValue() const { return valueOrError.index() == 0; }
    bool IsError() const { return !IsValue(); }

    const T& GetValueOrThrow() const {
        if (IsValue()) {
            return std::get<0>(valueOrError);
        }
        std::rethrow_exception(std::get<1>(valueOrError));
    }
};
```

Value Or Exception

Схема использования:

```
TValueOrError<int> GetResult(int param) {
    try {
        if (param == 0) {
            throw std::logic_error("param cant be zero");
        } else {
            return param / 2;
        }
    } catch (...) { return std::current_exception(); }
}

int main() {
    try {
        auto result = GetResult(0);
        std::cout << result.IsError() << std::endl;
        auto r = result.GetValueOrThrow();
        // use it ...
        std::cout << "result = " << r << std::endl;
    } catch (std::logic_error& e) {
        std::cout << "error: " << e.what() << std::endl;
    }
}
```

Value Or Exception

Что делать, если нужно прям по месту, не кидая исключения, записать исключение?

```
try {
    throw std::logic_error("logic error");
} catch (...) {
    TValueOrError<int> error = std::current_exception();
    std::cout << error.IsError() << std::endl;
}
```

Value Or Exception

Что делать, если нужно прям по месту, не кидая исключения, записать исключение?

```
try {
    throw std::logic_error("logic error");
} catch (...) {
    TValueOrError<int> error = std::current_exception();
    std::cout << error.IsError() << std::endl;
}
```

Чтобы так не делать, существует `std::make_exception_ptr`:

```
TValueOrError<int> error =
    std::make_exception_ptr(std::logic_error("logic error"));
std::cout << error.IsError() << std::endl;
```

noexcept

```
void f(int param) noexcept;
```

- Модификатор означает, что функция гарантированно не генерирует исключений.
- От этого может зависеть эффективность вызывающего кода.
- Допускается вызов из noexcept-функции других функций, которые noexcept не являются.
- В C++11 все функции освобождения памяти и все деструкторы неявно являются noexcept.

Гарантии безопасности исключений

1. Гарантия отсутствия исключений.
2. Строгая гарантия (исключения могут происходить, но все объекты остаются в согласованном и предсказуемом состоянии).
3. Базовая гарантия (исключения могут происходить, объекты остаются в согласованном состоянии, но не обязательно в предсказуемом).

Гарантии безопасности исключений

1. Гарантия отсутствия исключений.
2. Строгая гарантия (исключения могут происходить, но все объекты остаются в согласованном и предсказуемом состоянии).
3. Базовая гарантия (исключения могут происходить, объекты остаются в согласованном состоянии, но не обязательно в предсказуемом).

Пример: стек и операция `pop`.

- Согласованность означает соответствие size() и числа элементов при исключении.
- Предсказуемость означает, что при исключении число элементов в стеке не уменьшилось.

Assert

Проверяем предположения о данных объекта в программе, проверяем инварианты.

Assert

Проверяем предположения о данных объекта в программе, проверяем инварианты.

- assert времени компиляции

Программа не компилируется, если условие не выполняется.

```
static_assert(
    std::is_pointer<decltype(TCalcerPtr)>::value,
    "must be pointer"
);
```

Assert

- assert времени выполнения

```
#ifdef NDEBUG
#define assert(condition) ((void)0)
#else
#define assert(condition) /*implementation defined*/
#endif
```

А полезно ли это?