Проектирование больших систем на C++

Коноводов В. А.

кафедра математической кибернетики ВМК

Лекция 11
27.11.2018

Тестирование и зависимости

- Зависимости классов
- Как обеспечить изоляцию?

Основные подходы:

- Dummy. Передаются в тестируемые классы/методы/функции в виде параметра без поведения, внутри как правило с ним ничего не происходит
- Stub. Подменяется внешняя зависимость, игнорируются все данные, входящие в stub из тестируемого объекта.
- Fake. Замена легковесной реализацией.
- Mock. Объекты, которые имитируют поведение реальных. Полезно, когда настоящие объекты непрактично/невозможно вставлять в юнит-тест, но поведение нужно сохранить.

EXPECT_CALL: пример

```
EXPECT_CALL(db, func("string", _))
  .WillOnce(DoAll(SetArgReferee<1>("abc"), Return(false)));
```

- Метод `func` должен быть вызван 1 раз
- Ему будет передан аргумент `"string"`
- Что передадут в качестве второго аргумента — не важно
- Метод сделает второй аргумент равным `"abc"`
- Метод вернет `false`

Тестирование

1. Не стоит использовать `EXPECT_TRUE`
2. Добавляйте больше контекста
3. Код тестов — это тоже код

Тестирование

Тест упал, что делать дальше?

Основные подходы:

1. tracing и logging

```
cerr << "loading data: started" << endl;
```

2. strace и похожие утилиты
3. gdb и похожие отладчики
4. asan и другие санитайзеры

Debug

Пусть мы хотим залогировать все создания объекта класса A.

```
class A {
    A() { std::cerr << "A::A()" << std::endl;}
};
```

Класс B.

```
class B {
    B() { std::cerr << "B::B()" << std::endl;}
};
```

Некрасиво.

Debug: еще варианты

```
std::cerr << __FILE__ << " " << __LINE__ << std::endl;
```

Макросы, которые в момент компиляции раскрываются в имя файла и номер строчки.

strace

ОС — средство взаимодействия программы с внешним миром (файлы, соединения, ...)
strace is a diagnostic, debugging and instructional userspace utility for Linux. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state.
`strace ./you_binary`
Получаем все системные вызовы, предоставленные операционной системой

GDB

- **r**un – start the debugged program
- **l**ist – list specified function or line
- **b**reak – set breakpoint
- **cat**ch – set catchpoint (exception breakpoint)
- **i**nfo – show information about the debugged program
- **s**tep – step program, steps into functions
- **n**ext – step program, steps over function calls
- **s**tep**i**, **n**ext**i** – step by instructions, not lines of code
- **p**rint – evaluate expression
- e**x**amine – display contents of memory address
- **c**ontinue – continue running (after breakpoint)
- **k**ill – stop execution of the program
- **b**acktrace – print backtrace of stack frames
- **up**, **do**wn, **f**rame, **sel**ect-frame – select stack frame
- ...

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)

Удобно возвращать объекты std::optional 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;
            }
        }
    }
}
```