C/C++: Лекция 7

Воробьев Д.В

16.10.2020

Спецификации исключений до C++11

- Перечислить в throw типы, которые могут брошены
- Функция бросила тип не из списка вызов std::unexpected

Спецификации исключений до C++11

Тип из списка

```
class MyException {
    // реализация
};

void foo() throw(MyException) {
    // ок
    throw MyException();
}
```

Тип НЕ из списка

```
class MyException {
    // реализация
};

void foo() throw(MyException) {
    // std::unexpected
    throw 1;
}
```

std::unexpected

- Вызывается при выбрасывании типа не из списка
- Вызывает std::unexpected_handler
- std::unexpected_handler вызывает std::terminate

```
class MyException {
    // реализация
};

void foo() throw(MyException) {
    // std::unexpected
    throw 1;
}
```

Задаем свой handler

```
void handler() {
  std::cout << 1;
}

void foo() throw (double) {
  throw 1;
}

int main() {
  std::set_unexpected(handler);
  // 1
  foo();
  return 0;
}
```

Спецификации исключений с C++11

Введены:

- noexcept - спецификатор
- noexcept - оператор

Спецификатор noexcept

noexcept - не дает гарантии этапа компиляции на то, что функция не бросает ограничений

```
void bar() noexcept {
    throw 1;
}

int main() {
    bar();
}
```

Спецификатор noexcept

- noexcept - лишь метка компилятору для оптимизаций
- действительность отсутвия исключений на откуп проектировщику
- указание noexcept это ваш promise другому

Спецификатор noexcept

Перегрузку функций с разными категориями делать нельзя

```
// CE
void foo() noexcept;

void foo();
```

Спецификатор noexcept

Категория при наследовании не может ослабляться

```
struct Base {
    virtual void foo() noexcept;
};

struct Derived: Base {
    void foo(); // CE
};
```

Спецификатор noexcept

Категория при наследовании не может ослабляться

```
struct Base {
    virtual void foo();
};

struct Derived: Base {
    void foo() noexcept; // ok
};
```

Оператор noexcept

```
void foo();
void bar() noexcept;
struct X {
      ~X(){}
};

int main() {
    std::cout << noexcept(foo()) << std::endl;
    std::cout << noexcept(bar()) << std::endl;
    std::cout << noexcept(std::declval<X>().~X());
}
```

Условный noexcept

```
void bar() {
    throw 1;
}

void foo() noexcept( noexcept(bar(c)) ) {}
```

Исключения в конструкторах

```
void foo() {
    throw 1;
}
struct MyClass {
    int* x;
    MyClass() {
        x = new int(1);
        foo();
    }
    ~MyClass() {
        delete x;
        std::cout << 1;
    }
};

int main() {
    // деструктор не вызывается
    MyClass a;
}
```

Проблема утечки памяти при исключениях

Проблема: ptr не удаляется

```
void foo() {
    throw 1;
}

void Action() {
    int* ptr = new int(1);
    // действия
    // действия
    foo();
    // действия
    // действия
    delete ptr;
}
```

Проблема утечки памяти при исключениях

Решение: умные указатели

```
void foo() {
    throw 1;
}

void Action() {
    std::shared_ptr<int> ptr(new int(1));
    // при throw вызовется деструктор
    // локальных объектов
    foo();
}
```

Исключения в деструкторах

Не бросать. Причина: при раскрутке стека будет вызваны дестркуторы ранее созданных объектов снова выброс исключения 2 необработанных исключения

Исключения в деструкторах

C

C++11 деструкторы noexcept(true)

```
struct X {
      ~X(){}
};

int main() {
    // 1
    std::cout << noexcept(std::declval<X>().~X());
}
```

std::terminate

Вызывается в ситуации:

1. Есть непойманный exception
2. Бросается exception во время обработки

std::terminate : непойманный exception

```
class MyException {
    // реализациия
};

void foo() {
    throw MyException();
}

int main() {
    foo();
    return 0;
}
```

std::terminate : exception во время обработки

```
class MyException {
    // реализация
};

void foo() {
    throw MyException();
}

void bar() {
    throw MyException();
}
int main() {
    try {
        foo();
    } catch(MyException& e) {
        bar();
    }
    return 0;
}
```

std::set_terminate

```
// до C++11
std::terminate_handler set_terminate(std::terminate_handler f) throw();

// с C++11
std::terminate_handler set_terminate(std::terminate_handler f) noexcept;
```