

Исключения

Гусев Илья

Московский физико-технический институт

Москва, 2018

Содержание

- 1 Способы обработки ошибок
 - Коды возврата
 - assert
 - Исключения
- 2 Исключения
 - Синтаксис в C++
 - Коды возврата vs исключения
 - Исключения в конструкторах
 - Исключения в деструкторах
 - Как кидать и ловить?
 - Что такое throw; и catch (...)?
 - Исключения и полиморфизм
 - Спецификации исключений
 - SEH

Источники ошибок

- 1 Внешние источники - сбои в операционной системе, сбои в железе, ошибка соединения клиентской части с серверной.
- 2 Нехватка ресурсов: нехватка оперативной памяти, места на диске, ограничения на количество объектов.
- 3 Ошибки в логике программы.
- 4 Ошибки пользователя, например - неправильный ввод .
- 5 Классификация условная - куда отнести деление на 0?

Способы обработки ошибок

- ❶ Коды возврата — если функция возвращает `true` — все хорошо, если же `false` (или любое другое специальное значение) — то произошла ошибка. Иногда есть глобальная переменная с кодом ошибки (`GetLastError()`, `errno`).
- ❷ `Assert/abort` — роняем программу или подпрограмму.
- ❸ Исключения — прерывистый поток исполнения.

Коды возврата

```
int f1()
{
    // ...
    int rc = f2();
    if (rc == 0) {
        // ...
    } else {
        // код, обрабатывающий ошибку
    }
}
```

assert

```
#include <cassert>

int f1()
{
    // ...
    int rc = f2();
    assert(rc == 0);
    // если rc != 0, сюда мы не попадём
}
```

Исключения

```
void f1()
{
    try {
        // ...
        f2();
        // ...
    } catch (some_exception& e) {
        // код, обрабатывающий ошибку
    }
}
```

Синтаксис в C++

```
// Кинуть исключение:  
throw 123;  
throw 'c';  
throw std::logic_error("                ");  
  
// Поймать исключение:  
try {  
    // Код или вызовы функций, кидающие исключение  
} catch (int e) { // Вместо int - тип брошенного исключения  
    // Обработка ошибки  
}
```


Исключения

- ❶ Исключение принудительно вызывает код, который распознаёт ошибку и обрабатывает его. Необработанные исключения останавливают выполнение программы.
- ❷ Исключение переходит к точке в стеке вызовов, в которой можно обработать ошибку. Промежуточные функций пробрасывают исключение. Им не нужно общаться с другими функциями в стеке.
- ❸ Механизм очистки стека исключений уничтожает все объекты в области видимости в соответствии с чётко определёнными правилами, после того, как исключение брошено.
- ❹ Исключение обеспечивает четкое разделение между кодом, который обнаруживает ошибку и кодом, который обрабатывает ошибку.

Коды возврата vs исключения

- ❶ Исключения не накладывают на программиста обязанности проверять каждый вызов каждой функции, чтобы не потерять случайно возникшую ошибку.
- ❷ Если ошибка не обработана на текущем уровне — ничего страшного, ей займется более верхний. Возможно, ему уже будет известно, что с ней делать.
- ❸ Поддерживаются самим языком — нам не нужно модифицировать существующий код, чтобы научить его пробрасывать ошибку вверх этим новым способом.

Минусы исключений

- ❶ Исключения дают рваный поток выполнения. По коду нельзя сказать куда улетит брошенное нами исключение и что может вывалиться из того, что мы вызвали.
- ❷ Реальный путь исключения по стеку определяется не статическим описанием программы (сигнатуры методов с исключениями) а её динамической структурой (кто какие реализации вызывает).
- ❸ Исключения менее эффективны.
- ❹ При использовании исключений нужно очень аккуратно обращаться с динамической памятью, в идеале использовать Resource Acquisition Is Initialization (RAII).

Коды возврата vs исключения

```
void f(Number x, Number y)
{
    try {
        // ...
        Number sum  = x + y;
        Number diff = x - y;
        Number prod  = x * y;
        Number quot  = x / y;
        // ...
    }
    catch (Number::Overflow& exception) {
        // ...code that handles overflow...
    }
    catch (Number::Underflow& exception) {
        // ...code that handles underflow...
    }
    catch (Number::DivideByZero& exception) {
        // ...code that handles divide-by-zero...
    }
}
```

Коды возврата vs исключения

```
int f(Number x, Number y)
{
    // ...
    Number::ReturnCode rc;
    Number sum = x.add(y, rc);
    if (rc == Number::Overflow) {
        // ...code that handles overflow...
        return -1;
    } else if (rc == Number::Underflow) {
        // ...code that handles underflow...
        return -1;
    } else if (rc == Number::DivideByZero) {
        // ...code that handles divide-by-zero...
        return -1;
    }
}
```

Исключения в конструкторах

- 1 У конструктора нет возвращаемого значения.
- 2 Практически единственный способ сообщить об ошибке - кинуть исключение.
- 3 Память будет очищена, для членов класса - вызван деструктор.
- 4 Для самого объекта деструктор вызван не будет.
- 5 Члены класса должны сами за собой подчищать (RAII).

Исключения в деструкторах

- ❶ Исключения из деструктора практически никогда нельзя кидать.
Почему?
- ❷ Если в программе возникает исключение во время обработки другого исключения, происходит `terminate()`.
- ❸ При раскрутке стека вызываются деструкторы всех объектов на данном уровне стека.
- ❹ Вывод?
- ❺ Если используете какие-то функции внутри деструктора, и они кидают исключения - обрабатывайте исключения внутри деструктора.

Как кидать и ловить?

- ❶ Исключение может любой тип.
- ❷ Обычно тип исключения наследуют от `std::exception` или производных.
- ❸ Ловить можно по: значению, ссылке, указателю.
- ❹ Наиболее правильно - по ссылке.

Почему не по указателю?

```
MyException x;  
void f()  
{  
    MyException y;  
    try {  
        switch ((rand() >> 8) % 3) {  
            case 0: throw new MyException;  
            case 1: throw &x;  
            case 2: throw &y;  
        }  
    }  
    catch (MyException* p) {  
        // Здесь нам удалять p или нет?  
    }  
}
```

Что такое throw; и catch (...)?

```
void handleException()  
{  
    try {  
        throw; // пробросить текущее исключение дальше  
    }  
    catch (MyException& e) {  
        // обработка MyException  
    }  
    catch (YourException& e) {  
        // обработка YourException  
    }  
}  
  
void f()  
{  
    try {  
        // что-то, что кидает исключения  
    }  
    catch (...) { // ловит все исключения  
        handleException();  
    }  
}
```

Исключения и полиморфизм

```
class MyExceptionBase { };
class MyExceptionDerived : public MyExceptionBase { };
void f(MyExceptionBase& e)
{
    // ...
    throw e;
}
void g()
{
    MyExceptionDerived e;
    try {
        f(e);
    }
    catch (MyExceptionDerived& e) {
        // обработка MyExceptionDerived
    }
    catch (...) {
        // обработка остальных исключений
        // поток выполнения пойдёт сюда
    }
}
```

Спецификации исключений

```
void f() throw(int); // OK: function declaration
void (*fp)() throw (int); // OK: pointer to function declaration
void g(void pfa() throw(int)); // OK: pointer to function parameter
      declaration
typedef int (*pf)() throw(int); // Error: typedef declaration

void f() throw() // не бросает исключений
void f() throw(...) // может бросать исключения
```


Если функция кидает исключение, которое имеет тип, не указанный в спецификации, то вызывается `std::unexpected`. По умолчанию она вызывает `std::terminate`, но может быть заменена на пользовательскую функцию. Спецификация исключений в таком виде - довольно бесполезная и дорогая штука, планируют убрать из языка.


SEH


- 1 Windows-only
- 2 Позволяют ловить исключения процессора: access violation, illegal instruction, divide by zero.
- 3 Механизм, отличный от исключений C++
- 4 Один компилятор. Одна ОС. Не «чистый C++».


```
--try
{
    // guarded code
}
__except ( expression )
{
    // exception handler code
}
```

Полезные ссылки I


 C++ FAQ: Exceptions
<https://isocpp.org/wiki/faq/exceptions>

 Habr: Коды возврата vs исключения - битва за контроль ошибок
<https://habrahabr.ru/post/130611/>

 Habr: Коды возврата исключения
<https://habrahabr.ru/post/131212/>

 Habr: Обработка Segmentation Fault в C++ (про SEH)
<https://habrahabr.ru/post/131412/>

 MSDN: Обработка исключений C++
<https://msdn.microsoft.com/ru-ru/library/4t3saedz.aspx>

 CppReference: dynamic exception specification
http://en.cppreference.com/w/cpp/language/except_spec