УДК 004.438

АНАЛИЗ РАБОТЫ С ИСКЛЮЧЕНИЯМИ В РАЗЛИЧНЫХ ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

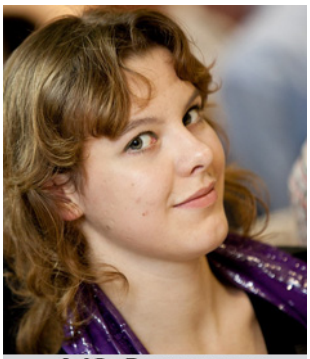
***Александра Юрьевна Волушкова**, магистр*
кафедра «Вычислительная математика и программирование»
Тел.:+7 926 153 5338, e-mail: a.volushkova@corp.mail.ru
МАИ (Московский авиационный институт)
http://www.mai.ru/unit/fpmf/806

В данной работе рассмотрены особенности обработки исключений в различных языках программирования. Созданы и реализованы алгоритмы оценки качества использования механизма исключений. Проведена оценка временных затрат этих механизмов для программ на компилируемых языках (C++, C#, Java, Eiffel).

Ключевые слова: исключение, exceptionhandling, языки программирования

Введение

Обработка исключений или исключительных ситуаций (англ. `exceptionhandling`) — механизм языков программирования, предназначенный для обработки ошибок времени выполнения и других возможных проблем (исключений), которые могут возникнуть при выполнении программы. При возникновении исключительной ситуации, управление передаётся некоторому заранее определённому обработчику. Так обработка ошибок передаётся на более высокий уровень и обеспечивается возможность так называемого нелокального выхода, т. е. передачи управления на некоторую «удалённую», возможно заранее неизвестную, точку программы через произвольное число вызовов функций. Основной недостаток исключений — в их невысокой скорости. Большинство современных языков программирования, такие как Ada, C++, D, Delphi, Objective-C, Java, JavaScript, Eiffel, OCaml, Ruby, Python, CommonLisp, SML, PHP, все языки платформы .NET и др. имеют встроенную поддержку обработки исключений. В этих языках, при возникновении исключения, происходит раскрутка стека вызовов до первого обработчика исключений подходящего типа, и управление передаётся обработчику.


А.Ю. Волушкова

Основной целью данной работы и является анализ аспектов работы механизма перехвата и обработки исключений, оценка «стоимости» этих механизмов для программы на различных языках программирования. За главный параметр качества взята производительность системы.

Для оценки качества использования механизма исключений разработаны тестовые алгоритмы для вычисления быстродействия механизма исключений в разных языках.

В качестве тестового алгоритма последовательной обработки исключений была выбрана сортировка простыми обменами, или сортировка пузырьком, имеющая сложность $O(n^2)$.

В качестве тестового алгоритма вложенной обработки исключений была выбрана быстрая сортировка.

Быстрая сортировка (англ. `quicksort`) — широко известный алгоритм сортировки, разработанный английским учёным Чарльзом Хоаром. Самый быстрый из известных универсальных алгоритмов сортировки массивов (в среднем O(n*log n) обменов при упорядочении `n` элементов).

Для получения статистических данных, используется циклический вызов функций от 10000 раз, но в случае с пузырьковой сортировкой с исключениями – 100 раз, т.к. выполнение одной итерации занимает слишком много времени.

Результатом тестирования являются следующие данные.

ArrayLength – количество элементов в сортируемом массиве;

IterationsCount – количество выполняемых итераций (сортировок);

FullTime – общее время выполнения всех итераций (сортировок);

AverageTime – среднее время выполнения одной итерации (сортировки);

TotalExceptionsCount – общее количество бросаемых исключений (количество исключений во время выполнения всех сортировок данного типа);

AverageExceptionsCount - среднее количество бросаемых исключений (количество исключений за одну сортировку);

AlgorithmCorrect – корректность алгоритма (проверяется, что на входе имеется неотсортированный массив, а на выходе - отсортированный).

Время выводится в миллисекундах, что равно 1/1000 секунды.

Пример полученного результата для массива длиной 200 элементов:

```
Test Name: Bubble sort
Array Length: 1000
Iteration Count: 10000
Full Time: 29328 msec
Average Time: 2,9328 msec
Total Exceptions Count: 0
Average Exceptions Count: 0
Algorithm Correct: True
```

```
Test Name: Bubble sort with exceptions
Array Length: 1000
Iteration Count: 100
Full Time: 728328 msec
Average Time: 7283,28 msec
Total Exceptions Count: 25379500
Average Exceptions Count: 253795
Algorithm Correct: True

Test Name: Quick sort
Array Length: 1000
Iteration Count: 10000
Full Time: 1187 msec
Average Time: 0,1187 msec
Total Exceptions Count: 0
Average Exceptions Count: 0
Algorithm Correct: True

Test Name: Quick sort with exceptions
Array Length: 1000
Iteration Count: 10000
Full Time: 247593 msec
Average Time: 24,7593 msec
Total Exceptions Count: 6760000
Average Exceptions Count: 676
Algorithm Correct: True
```

Из данных результатов можно получить среднее количество исключений, обрабатываемых за одну миллисекунду, для этого делим среднее число исключений на среднее время выполнения тестового алгоритма с исключениями минус среднее время выполнения тестового алгоритма без исключений (т.е. мы получаем чистое время на обработку исключений без времени работы самого алгоритма):

`676 / (24,7593 - 2,9328) = 30,97152543926` – в случае последовательной обработки исключений;

`253795 / (7283,28 - 0,1187) = 34,84681850998` – в случае вложенной обработки исключений.

Также можно заметить, что выполнение 1 итерации сортировки пузырьком с исключениями по времени равнозначно выполнению около 2500 итераций без исключений, а выполнение 1 итерации быстрой сортировки с исключениями по времени равнозначно выполнению около 210 итераций без исключений.

Тесты проводятся на компьютере с параметрами:

Процессор – Intel (r) Core (tm) 2 CPU 6300 @ 1,86GHz;

ОЗУ – 2ГБ;

Проведём анализ работы с исключениями в компилируемых языках и рассмотрим *Особенности обработки исключений в C++,в C#, Eiffel, Java* .

В языке C++ реализован специальный механизм исключительных ситуаций [1]. Исключительная ситуация возникает при выполнении оператора `throw` . В качестве аргумента `throw` задаётся любое значение. Это может быть значение одного из встроенных типов (число, строка символов и т.п.) или объект любого определённого в программе класса.

При возникновении исключительной ситуации выполнение текущей функции или метода немедленно прекращается, созданные к этому моменту автоматические переменные уничтожаются, и управление передаётся в точку, откуда была вызвана текущая функция или метод. В точке возврата создаётся та же самая исключительная ситуация, прекращается выполнение текущей функции или метода, уничтожаются автоматические переменные, и управление передаётся в точку, откуда была вызвана эта функция или метод. Происходит своего рода откат всех вызовов до тех пор, пока не завершится функция `main` и, соответственно, вся программа.

В C++ есть возможность определить для исключений иерархию классов – по классу на каждый тип исключительной ситуации.

Чтобы облегчить обработку ошибок и сделать запись о них более наглядной, описания методов и функций можно дополнить информацией, какого типа исключительные ситуации они могут создавать:

```
class Database
{
public :
Open(const char*serverName) throw ConnectDbException;
};
```

Такое описание говорит о том, что метод `Open` класса `Database` может создать исключительную ситуацию типа `ConnectDbException`. Соответственно, при использовании этого метода желательно предусмотреть обработку возможной исключительной ситуации.

Если исключительная ситуация возникла в конструкторе объекта, считается, что объект сформирован не полностью, и деструктор для него вызван не будет.

Язык C# наследовал схему исключений языка C++, внеся в неё свои коррективы [2]. Рассмотрим схему подробнее и начнём с синтаксиса конструкции `try-catch-finally`:

```
try
{…}
```

```
catch (T1 e1)
{…}
…
catch(Tk ek)
{…}
finally
{…}
```

Всюду в тексте модуля, где синтаксически допускается использование блока, этот блок можно сделать охраняемым, добавив ключевое слово `try`. Вслед за `try`-блоком могут следовать catch-блоки, называемые блоками-обработчиками исключительных ситуаций, их может быть несколько, они могут и отсутствовать. Завершает эту последовательность `finally`-блок – блок финализации, который также может отсутствовать. Вся эта конструкция может быть вложенной – в состав `try`-блока может входить конструкция `try-catch-finally`.

В рассматриваемой нами модели исключения являются объектами, класс которых представляет собой наследника класса `Exception`. Этот класс и многочисленные его наследники является частью библиотеки FCL, хотя и разбросаны по разным пространствам имён. Каждый класс задаёт определённый тип исключения в соответствии с классификацией, принятой в Framework.Net.

При выполнении оператора `throw` создается объект `te`, класс `TE` которого характеризует текущее исключение, а поля содержат информацию о возникшей исключительной ситуации. Выполнение оператора `throw` приводит к тому, что нормальный процесс вычислений на этом прекращается. Если это происходит в охраняемом `try`-блоке, то начинается этап «захвата» исключения одним из обработчиков исключений.

Класс `T`, указанный в заголовке `catch`-блока, должен принадлежать классам исключений. Блок `catch` с формальным аргументом `e` класса `T` потенциально способен захватить текущее исключение `te` класса `TE`, если и только если объект `te` совместим по присваиванию с объектом `e`. Другими словами, потенциальная способность захвата означает допустимость присваивания `e = te`, что возможно, когда класс `TE` является потомком класса `T`. Обработчик, класс `T` которого является классом `Exception`, является универсальным обработчиком, потенциально он способен захватить любое исключение, поскольку все они являются его потомками.

Потенциальных захватчиков может быть много, исключение захватывает лишь один – тот из них, кто стоит первым в списке проверки. Когда же будет исчерпаны списки вложенных блоков, а потенциальный захватчик не будет найден, то произойдет подъем по стеку вызовов.

Исключения в Eiffel есть, но они особенные. Скорее это похоже на механизм сигналов в Unix, хотя при порождении исключения происходит раскрутка стека до первого обработчика, само исключение – это не объект [3]. Скорее это набор неких атрибутов: код исключения и связанное с ним сообщение.

Обработчик исключения в методе может быть только один – он пишется в специальной секции `rescue`. Обработчик может содержать обычные инструкции и специальную команду `retry`.

Использование `retry` предписывает выполнить метод повторно с самого начала (при этом все промежуточные значения, полученные при предшествующем запуске, остаются на своих местах). Если команды `retry` нет, то по завершении секции `rescue` исключение выбрасывается наружу.

К механизму обработки исключений в Java имеют отношение 5 ключевых слов: — `try, catch, throw, throws` и `finally`. Схема работы этого механизма следующая. Вы пытаетесь `try` выполнить блок кода, и если при этом возникает ошибка, система возбуждает `throw` исключение, которое в зависимости от его типа вы можете перехватить `catch` или передать умалчиваемому `finally` обработчику [4].

В вершине иерархии исключений стоит класс `Throwable`. Каждый из типов исключений является подклассом класса `Throwable`. Два непосредственных наследника класса `Throwable` делят иерархию подклассов исключений на две различные ветви. Один из них — класс `Exception` — используется для описания исключительных ситуации, которые должны перехватываться программным кодом пользователя. Другая ветвь дерева подклассов `Throwable` — класс `Error`, который предназначен для описания исключительных ситуаций, которые при обычных условиях не должны перехватываться в пользовательской программе.

Только подклассы класса `Throwable` могут быть возбуждены или перехвачены. Простые типы — `int, char` и т.п., а также классы, не являющиеся подклассами `Throwable`, например, `String` и `Object`, использоваться в качестве исключений не могут. Наиболее общий путь для использования исключений — создание своих собственных подклассов класса `Exception`.

Ниже приведем результаты выполненных тестовых алгоритмов.

Для сортировки на компилируемых языках был выбран массив из 1000 элементов.

Таблица 1.

Пузырёк:

	C++	C#	Java	Eiffel
Кол-во итераций	10000	10000	10000	10000
Общее время (мсек)	28407	28953	45610	170704
Среднее время (мсек)	2,84	2,8953	4,561	170,704

Таблица 2.

Пузырёк с исключениями

	C++	C#	Java	Eiffel
Кол-во итераций	100	100	100	100
Общее время (мсек)	77593	697093	3781	31528,2
Среднее время (мсек)	775,93	6970,93	37,81	315,282
Общее кол-во исключений	24422300	25225600	25893800	25362700
Среднее кол-во исключений	244223	252256	258938	253627
Исключений за мсек	315,9	36,2	7787,84	1754,26

Таблица 3.

Быстрая сортировка

	C++	C#	Java	Eiffel
Кол-во итераций	10000	10000	10000	10000
Общее время (мсек)	844	1140	1531	45907
Среднее время (мсек)	0,08	0,114	0,1531	4,5907

Таблица 4.

Быстрая сортировка с исключениями

	C++	C#	Java	Eiffel
Кол-во итераций	10000	10000	10000	10000
Общее время (мсек)	23531	236031	36078	60562
Среднее время (мсек)	2,35	23,6031	3,6078	6,0562
Общее кол-во исключений	6770000	6710000	6590000	11640000
Среднее кол-во исключений	677	671	659	1164
Исключений за мсек	298,2	28,56	190,75	794,26

Исходя из полученных результатов, можно сделать вывод, что вложенная обработка исключений в языках C++ и C# больше нагружает систему при выполнении, чем обработка исключений на месте их возникновения, но не значительно. В языке Java обработка исключений на месте их возникновения практически не влияет на скорость работы программы (производительность превосходит язык C++ в 20 раз и C# в 200 раз), а при подъеме по стеку вызовов значительно замедляет программу (здесь результат уже сравним с языком C++).

Отдельного разговора заслуживает язык Eiffel: в нём обработка исключений устроена совершенно непохожим на все остальные механизмы образом. По сути, многие конструкции языка созданы для предотвращения возникновения исключений. Сама программа выполнялась дольше других испытуемых, но, тем не менее, мы видим отличный результат: 1750 исключений за миллисекунду при обработке исключений в месте их возникновения и почти 800 при подъеме по стеку вызовов.

Заключение

Автор считает, что в данной работе новыми являются следующие положения и результаты:

- набор алгоритмов, дающий оценку производительности программ, написанных с использованием исключений (обрабатываемых в месте их возникновения и с переходом выше по стеку вызова функций) и без.
- анализ особенностей механизмов перехвата и обработки исключений и оценка «стоимости» этих механизмов для программ на компилируемых языках (C++, C#, Java, Eiffel).

В целом, по результатам тестов можно сделать вывод, что во всех языках программирования исключения снижают производительность, особенно если их обработка передаётся выше по стеку вызова функций. Следовательно, следует использовать исключения максимально разумно – там, где нельзя их избежать и там, где это диктуется архитектурой приложения, но при этом и не игнорировать критичные исключения.

Литература

1. Бьёрн Страуструп Язык программирования C++: специальное издание – СПб.: Бином, Невский Диалект, 2004. – 1054 с.

2. *Кристиан Нейгел, Билл Ивьен, Джей Глинн, Карли Уотсон, Морган Скиннер.* C# 4.0 и платформа .NET 4 для профессионалов. – М.: Вильямс, 2011. – 1440 с.

3. *Бертран Мейер*. Объектно-ориентированное конструирование программных систем. – М.: Русская Редакция, 2005. – 534 с.

4. *Хорстманн, Кей С., Корнел, Гари.* Java 2. Библиотека профессионала. Том 2: Тонкости программирования. – 8-е изд. – М.: Вильямс, 2009. – 992 с.

Analysis of exception handling in different programming languages

Alexsandra Yur'evna Volushkova, master MAI (Moscow Aviation Institute)
MAI (Moscow Aviation Institute)
Department of Calculus Mathematics and Programming

This work considers peculiarities of exception handling in different programming languages. Algorithms to evaluate the usage of exception mechanisms were created and implemented. Time costs of these mechanisms in compliable languages (C++, C#, Java, Eiffel) were measured and analyzed.

Key words: Exception, exception handling, programming languages