Обработка исключительных ситуаций в C#

Введение в исключения

Рассмотрим, как в языке C# на наиболее простом уровне может осуществляться перехват и обработка исключений. Данные приемы могут значительно повысить надежность и область применимости программных кодов.

Обработка ошибок в C# в определенном смысле напоминает синтаксическую конструкцию, подобную условному оператору. Но, если в условном операторе проверяется логическое выражение (условие), то при обработке исключений роль «условия» играет ошибка, которая может возникнуть, а может не возникнуть.

Что же такое исключение и что с ним можно делать? Дело в том, что при выполнении программы могут возникать ошибки. Речь не идет об ошибках, связанных с неправильно написанным программным кодом. Речь об ошибках, которые могут возникнуть при выполнении программы. Например, программа выдает пользователю запрос на ввод числа. Но нет никакой гарантии, что пользователь введет именно число. В таком случае, когда программа попытается преобразовать введенное пользователем значение в число, может возникнуть ошибка. То есть данная ситуация полностью зависит от действий пользователя. Еще пример. При вычислении выражений могут возникнуть ошибки, например, переполнение, исчезновение порядка или деление на ноль.

Программист может предусмотреть реакцию программы на ошибочные ситуации. В C# есть механизм, который позволяет обрабатывать подобные ошибки и таким образом избегать аварийного завершения программы. Он так и называется: механизм обработки исключительных ситуаций (исключений).

Если в процессе вычислений возникла ошибка, система сигнализирует об этом с помощью специального действия, называемого выбрасыванием (генерированием) исключения. Каждому типу ошибки соответствует свое исключение. Поскольку C# — язык объектно-ориентированный, исключения являются классами, которые имеют общего предка — класс Exception, определенный в пространстве имен System.

Например, при делении на ноль будет выброшено (сгенерировано) исключение с длинным, но понятным именем DivideByZeroException, при недостатке памяти — исключение OutOfMemoryException, при переполнении — исключение OverflowException.

ПРИМЕЧАНИЕ. Стандартных исключений очень много, тем не менее, программист может создавать и собственные исключения на основе класса Exception.

Программист может задать способ обработки исключения в специальном блоке кода, начинающемся с ключевого слова catch («перехватить»), который будет автоматически выполнен при возникновении соответствующей исключительной ситуации.

Внутри блока можно, например, вывести предупреждающее сообщение или скорректировать значения величин и продолжить выполнение программы. Если этот блок не задан, система выполнит действия по умолчанию, которые обычно заключаются в выводе диагностического сообщения и нормальном завершении программы.

Процессом выбрасывания исключений, возникающих при переполнении, можно управлять. Для этого служат ключевые слова checked и unchecked. Слово checked включает проверку переполнения, слово unchecked выключает. При выключенной проверке исключения, связанные с переполнением, не генерируются, а результат операции усекается. Проверку переполнения можно реализовать для отдельного выражения или для целого блока операторов, например:

```
a = checked (b + c); // для выражения

unchecked {     // для блока операторов

a = b + c;

}
```

Проверка не распространяется на функции, вызванные в блоке. Если проверка переполнения включена, говорят, что вычисления выполняются в проверяемом контексте, если выключена — в непроверяемом. Проверку переполнения выключают в случаях, когда усечение результата операции необходимо в соответствии с алгоритмом.

Можно задать проверку переполнения во всей программе с помощью ключа компилятора /checked, это полезно при отладке программы. Поскольку подобная проверка несколько замедляет работу, в готовой программе этот режим обычно не используется.

Обработка исключительных ситуаций

В языке C# есть операторы, позволяющие обнаруживать и обрабатывать ошибки (исключительные ситуации), возникающие в процессе выполнения программы. Рассмотрим механизм обработки исключений более подробно.

Исключительная ситуация, или ***исключение,*** — это возникновение аварийного события, которое может порождаться некорректным использованием аппаратуры или неправильной работой программы, например делением на ноль или переполнением. Обычно эти события приводят к завершению программы с системным сообщением об ошибке. C# дает программисту возможность восстановить работоспособность программы и продолжить ее выполнение.

Исключения C# не поддерживают обработку асинхронных событий, таких как ошибки оборудования или прерывания, например нажатие клавиш Ctrl+C. Механизм исключений предназначен только для событий, которые могут произойти в результате работы самой программы и указываются явным образом. Исключения возникают тогда, когда некоторая часть программы не смогла сделать то, что от нее требовалось. При этом другая часть программы может попытаться сделать что-нибудь иное.

Исключения позволяют логически разделить вычислительный процесс на две части — обнаружение аварийной ситуации и ее обработка. Это важно не только для лучшей структуризации программы. Главное то, что функция, обнаружившая ошибку, может не знать, что предпринимать для ее исправления, а использующий эту функцию код может

знать, что делать, но не уметь определить место возникновения. Это особенно актуально при использовании библиотечных функций и программ, состоящих из многих модулей.

Другое достоинство исключений состоит в том, что для передачи информации об ошибке в вызывающую функцию не требуется применять возвращаемое значение или параметры, поэтому заголовки функций не разрастаются.

ПРИМЕЧАНИЕ. В принципе, ничто не мешает рассматривать в качестве исключений не только ошибки, но и нормальные ситуации, возникающие при обработке данных, но это не имеет преимуществ перед другими решениями, не улучшает структуру программы и не делает ее понятнее.

Исключения генерирует либо среда выполнения, либо программист с помощью оператора throw. В табл. 4.1 приведены наиболее часто используемые стандартные исключения, генерируемые средой. Они определены в пространстве имен System. Все они являются потомками класса Exception, а точнее, потомками его потомка SystemException.

Исключения обнаруживаются и обрабатываются в операторе try.

Таблица 4.1. Часто используемые стандартные исключения

Имя	Описание
ArithmeticException	Ошибка в арифметических операциях или преобразованиях (является предком DivideBeZeroException и OverFlowException)
ArrayTypeMismatchException	Попытка сохранения в массиве элемента несовместимого типа
DivideByZeroException	Попытка деления на ноль
FormatException	Попытка передать в метод аргумент неверного формата
IndexOutOfRangeException	Индекс массива выходит за границы диапазона
InvalidCastException	Ошибка преобразования типа
OutOfMemoryException	Недостаточно памяти для создания нового объекта
OverFlowException	Переполнение при выполнении арифметических операций
StackOverFlowException	Переполнение стека

Оператор try

Оператор try содержит три части:

- контролируемый блок — составной оператор, предваряемый ключевым словом try. В контролируемый блок включаются потенциально опасные операторы программы. Все функции, прямо или косвенно вызываемые из блока, также считаются ему принадлежащими;

- один или несколько обработчиков исключений — блоков catch, в которых описывается, как обрабатываются ошибки различных типов;

□ блок завершения finally выполняется независимо от того, возникла ошибка в контролируемом блоке или нет.

Синтаксис оператора try:

```
try блок [ блоки catch ] [ блок finally ]
```

Отсутствовать могут либо блоки catch, либо блок finally, но не оба одновременно.

Рассмотрим, каким образом реализуется обработка исключительных ситуаций.

1. Обработка исключения начинается с появления ошибки. Функция или операция, в которой возникла ошибка, генерирует исключение. Как правило, исключение генерируется не непосредственно в блоке try, а в функциях, прямо или косвенно в него вложенных.

2. Выполнение текущего блока прекращается, отыскивается соответствующий обработчик исключения, и ему передается управление.

3. Выполняется блок finally, если он присутствует (этот блок выполняется и в том случае, если ошибка не возникла).

4. Если обработчик не найден, вызывается стандартный обработчик исключения. Его действия зависят от конфигурации среды. Обычно он выводит на экран окно с информацией об исключении и завершает текущий процесс.

ПРИМЕЧАНИЕ. Подобные окна не предназначены для пользователей программы, поэтому все исключения, которые могут возникнуть в программе, должны быть перехвачены и обработаны.

Обработчики исключений должны располагаться непосредственно за блоком try.

Они начинаются с ключевого слова catch, за которым в скобках следует тип обрабатываемого исключения. Можно записать один или несколько обработчиков в соответствии с типами обрабатываемых исключений. Блоки catch просматриваются в том порядке, в котором они записаны, пока не будет найден соответствующий типу выброшенного исключения.

Синтаксис обработчиков напоминает определение функции с одним параметром — типом исключения. Существуют три формы записи:

```
catch ( тип имя ){ ... /* тело обработчика */ }

catch( тип )    {.../* тело обработчика */ }

catch { ... /* тело обработчика */ }
```

Первая форма применяется, когда имя параметра используется в теле обработчика для выполнения каких-либо действий, например вывода информации об исключении.

Вторая форма не предполагает использования информации об исключении, играет роль только его тип.

Третья форма применяется для перехвата всех исключений. Так как обработчики просматриваются в том порядке, в котором они записаны, обработчик третьего типа (он может быть только один) следует помещать после всех остальных. Пример:

```
try {
...   // Контролируемый блок
}
catch ( OverflowException e ) {
//    Обработка    исключений    класса    OverflowException
(переполнение)
}
catch ( DivideByZeroException ) {
...//  Обработка  исключений  класса  DivideByZeroException
(деление на 0)
}
catch {
// Обработка всех остальных исключений
}
```

Если исключение в контролируемом блоке не возникло, все обработчики пропускаются.

В любом случае, произошло исключение или нет, управление передается в блок завершения finally (если он существует), а затем — первому оператору, находящемуся непосредственно за оператором try. В завершающем блоке обычно записываются операторы, которые необходимо выполнить независимо от того, возникло исключение или нет, например, закрытие файлов, с которыми выполнялась работа в контролируемом блоке, или вывод информации.

В листинге 4.9 приведена программа, вычисляющая силу тока по заданным напряжению и сопротивлению. Поскольку вероятность неверного набора вещественного числа довольно высока, оператор ввода включен в контролируемый блок.

ПРИМЕЧАНИЕ. Исключение, связанное с делением на ноль, для вещественных значений возникнуть не может, поэтому не проверяется. При делении на ноль будет выдан результат «бесконечность».

Листинг 4.9. Использование исключений для проверки ввода

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication15
{
    class Program
    {
        static void Main(string[] args)
        {
            string buf;
```

```
            double u, i, r;
        try
        {
            Console.WriteLine( "Введите напряжение:" );
            u= double.Parse( Console.ReadLine() );
            Console.WriteLine( "Введите сопротивление:" );
            r = double.Parse( Console.ReadLine() );
            i = u / r;
            Console.WriteLine( "Сила тока - " + i );
        }
        catch ( FormatException )
        {
            Console.WriteLine ( "Неверный формат ввода!" );
        }
        catch      // общий случай
        {
            Console.WriteLine ( "Неопознанное исключение" );
        }
        Console.ReadKey();
       }
    }
}
```

Итак, для обработки исключений используется конструкция `try-catch`. Программный код, при выполнении которого может возникнуть ошибка (***контролируемый код***), помещается в `try`-блок. После этого блока размещается `catch`-блок, в котором размещается программный код, предназначенный для обработки исключения. Если при выполнении кода в `try`-блоке ошибка не возникает, то `catch`-блок игнорируется. Если при выполнении кода в `try`-блоке возникает ошибка, то выполнение команд в блоке `try` прекращается и начинают выполняться команды в блоке `catch`.

Общий вид конструкции представлен ниже:

```
try {
        // контролируемый код
}
catch {
        // код для обработки исключений
}
```

Операторы `try` могут многократно вкладываться друг в друга. Исключение, которое возникло во внутреннем блоке `try` и не было перехвачено соответствующим блоком `catch`, передается на верхний уровень, где продолжается поиск подходящего обработчика. Этот процесс называется распространением исключения.

Распространение исключений предоставляет программисту интересные возможности. Например, если на внутреннем уровне недостаточно информации для того, чтобы провести полную обработку ошибки, можно выполнить частичную обработку и сгенерировать исключение повторно, чтобы оно было обработано на верхнем уровне. Генерация исключения выполняется с помощью оператора `throw`.

Оператор throw

До сих пор мы рассматривали исключения, которые генерирует среда выполнения C#, но это может сделать и сам программист. Для генерации исключения используется оператор throw с параметром, определяющим вид исключения. Параметр должен быть объектом, порожденным от стандартного класса System.Exception. Этот объект используется для передачи информации об исключении его обработчику.

Оператор throw употребляется либо с параметром, либо без него:

```
throw [ выражение ];
```

Форма без параметра применяется только внутри блока catch для повторной генерации исключения. Тип выражения, стоящего после throw, определяет тип исключения, например:

```
throw new DivideByZeroException();
```

Здесь после слова throw записано выражение, создающее объект стандартного класса «ошибка при делении на 0» с помощью операции new.

При генерации исключения выполнение текущего блока прекращается и происходит поиск соответствующего обработчика с передачей ему управления. Обработчик считается найденным, если тип объекта, указанного после throw, либо тот же, что задан в параметре catch, либо является производным от него.

Рассмотрим пример, приведенный в спецификации C#:

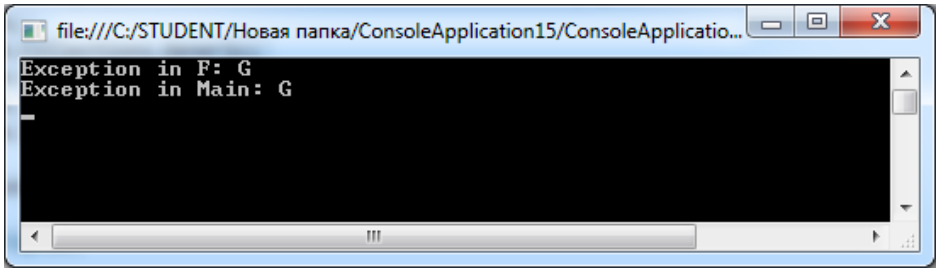
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication15
{
    class Program
    {
        static void F()
        {
            try {
                G();// функция, в которой-может произойти исключение
            }
            catch ( Exception e ) {
                Console.WriteLine( "Exception in F: " + e.Message );
                e = new Exception( "E" );
                throw;      // повторная,генерация исключения
            }
        }
        static void G()
        {
            throw new Exception("G");    // моделирование исключительной ситуации
        }

        static void Main(string[] args)
        {
            try {
                F();
```

```
            }
            catch ( Exception e){
                Console.WriteLine( "Exception in Main: " + e.Message );
            }
            Console.ReadKey();
        }
    }
}
```

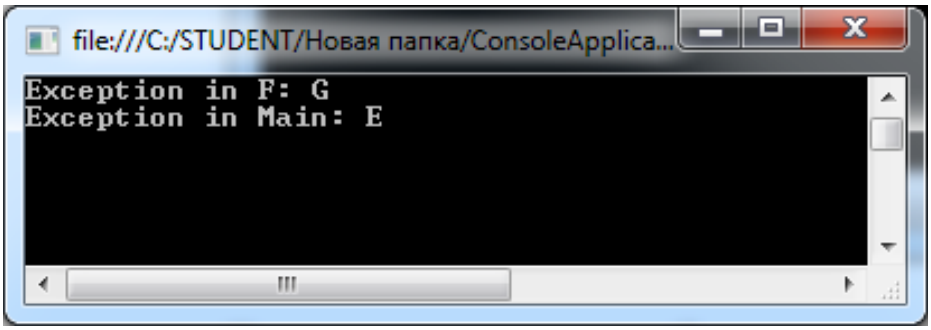
В методе F выполняется промежуточная обработка исключения, которая заключается в том, что на консоль выводится поле Message перехваченного объекта e (об элементах класса Exception рассказывается в следующем разделе). После этого исключение генерируется заново. Несмотря на то, что в обработчике исключения создается новый объект класса Exception с измененной строкой информации, передаваемой в исключении, выбрасывается не этот объект, а тот, который был перехвачен обработчиком, поэтому результат работы программы следующий:



Заменим оператор throw таким оператором:

```
throw e;
```

В этом случае в обработчике будет выброшено исключение, созданное в предыдущем операторе, и вывод программы изменится:



Класс Exception

Класс Exception содержит несколько полезных свойств, с помощью которых можно получить информацию об исключении. Они перечислены в табл. 4.2.

Таблица 4.2. Свойства класса System.Exception

Свойство	Описание
HelpLink	URL файла справки с описанием ошибки
Message	Текстовое описание ошибки. Устанавливается при создании объекта. Свойство доступно только для чтения
Source	Имя объекта или приложения, которое сгенерировало ошибку
StackTrace	Последовательность вызовов, которые привели к возникновению ошибки. Свойство доступно только для чтения
InnerException	Содержит ссылку на исключение, послужившее причиной генерации текущего исключения
TargetSite	Метод, выбросивший исключение

Операторы checked и unchecked

Как уже упоминалось, процессом генерации исключений, возникающих при переполнении, можно управлять с помощью ключевых слов `checked` и `unchecked`, которые употребляются как операции, если они используются в выражениях, и как операторы, если они предваряют блок, например:

```
a = checked (b + c);       // для выражения (проверка включена)

unchecked  {     // для блока операторов (проверка выключена)
      a = b + c:
}
```

Проверка не распространяется на функции, вызванные в блоке.