Основы платформы Microsoft .NET

Тема:

Обработка ошибок

Понятие исключения

Рассмотрим проблему, связанную с контролем и обработкой ошибок, возникающих в ходе выполнения программы. Предположим, что необходимо написать метод, который загружает в память графическое изображение, хранящееся в файле. В качестве параметра методу передается путь к файлу и область памяти, куда необходимо загрузить изображение. Во время работы данного метода может возникнуть ряд ошибок, например, файл может не существовать, недостаточно места для загрузки файла и т.д. Обо всех этих ошибках необходимо сообщить пользователю разрабатываемого метода каким-либо способом. Одним из вариантов организации такого сообщения является широко используемый в практике способ, при котором в качестве кода завершения используется возвращаемое значение функции (например, 0 – нет ошибок, 1 – файл не найден и т.д.)

```
int OpenFile (string path, Buffer buf) {…}
```

Программа, вызывающая этот метод, должна обрабатывать некоторым образом ошибочные ситуации, например:

```
int result = OpenFile ("c:\\test.gif", buffer);
if (res == 1) {
  Console.WriteLine ("Файл не найден");
}
else if (res == 2) {
  Console.WriteLine ("Недостаточно места для загрузки файла")
}
…
```

У данного подхода есть ряд недостатков. Во-первых, код, обрабатывающий ошибочные ситуации, довольно громоздок и нечитабельный, во-вторых, само значение ошибки не несет никакой дополнительной информации, причем сами значения

придумываются самим программистом и, например, другой метод может возвращать 1 в случае успешного выполнения, что может привести к путанице и ошибкам.

В качестве решения обозначенной проблемы в Microsoft .NET предлагается использовать широко используемы в практике программирования более профессиональный способ обработки ошибочных ситуаций - *механизм исключений*. Под *исключением* или *исключительной ситуацией* (*exception*) понимается возникновение некоторого особого события в программе. Как правило, под исключением подразумевается обнаружение ошибки, хотя механизм исключений может быть использован для обработки и других возникающих особых ситуаций в процессе выполнения программы.

Microsoft .NET Framework содержит набор классов, которые позволяют сохранить информацию о возникающих исключениях, а также предоставляет механизм передачи и перехвата объектов этих классов.

Обработка исключений

Рассмотрим, каким образом реализована работа с исключениями в Microsoft .NET Framework. Перехват и обработка исключений осуществляется с помощью блоков ***try…catch***, общий формат использования которых является следующим:

```
try {
  // код программы, который может вызвать исключение
  …
}
catch (<Класс_исключения> переменная) {
  // код, выполняющийся в случае возникновения исключения
  …
}
```

Таким образом, код разделен на две части.

Блок try

Первая часть заключена в блоке ***try*** и является обычным кодом программы, в котором в результате работы может возникнуть исключительная ситуация (например, ошибка). Если никаких ошибок не произошло, то после выполнения блока ***try*** управление передастся на строку, следующую за последним блоком ***catch*** (программный код блоков ***catch*** пропускается).

Код, который необходимо выполнить в случае возникновения исключения, размещается в одном или нескольких блоках ***catch***, связанных с блоком ***try***.

Блок catch

Вторая часть кода помещена в блок ***catch***, который принимает в качестве параметра объект класса *Exception* или класса, являющегося его потомком. Как только при выполнении очередного оператора блока ***try*** происходит исключение, то управление передается в блок ***catch***, и оставшиеся операторы в блоке ***try*** выполнены не будут.

У блока ***try*** должен быть как минимум один блок ***catch***. Если выполнение программного кода из блока ***try*** не привело к возникновению исключений, то программный код блоков ***catch*** пропускается и не исполняется.

То, каким образом в случае возникновения исключения выбирается один из нескольких блоков ***catch***, связанных с одним блоком ***try***, будет рассмотрено ниже.

Блок finally

Блок ***finally*** содержит код, который будет гарантированно исполнен вне зависимости от того, возникло исключение или нет. Например, если в блоке ***try*** был открыт файл, то независимо от того, возникнет исключение или нет, файл все равно нужно в конце работы закрыть. В таком случае, код, отвечающий за закрытие файла целесообразно разместить в блоке ***finally***:

```
void ReadData(String path) {
  FileStream fs = null;
  try {
    fs = new FileStream(path, FileMode.Open);
    // Обработка данных из файла, которая может
    // вызвать исключение
    ...
  }
  catch (OverflowException) {
    // Обработка исключительной ситуации
  }
  finally {
    // Обязательное закрытие файла
    if (fs != null) {
      fs.Close();
      fs = null;
    }
  }
}
```

Пример обработки исключений

Рассмотрим пример обработки исключений. Ниже приведен код программы, которая запрашивает у пользователя два вещественных числа и затем выводит результат деления первого числа на второе.

```
using System;

class ExceptionDemo {
  public static void Main ()   {
```

```
      try {
        Console.WriteLine ("Введите первое число:");
        int n = int.Parse (Console.ReadLine());
        Console.WriteLine ("Введите второе число:");
        int m = int.Parse (Console.ReadLine());
        int res = n / m;
        Console.WriteLine ("Res={0}", res);
      }
      catch (OverflowException e) {
        Console.WriteLine (e);
      }
    }
  }
```

Подготовьте эту программу и выполните несколько экспериментов. Например, введите вместо числа символ или в качестве второго числа 0. Так как делить на ноль нельзя, то программа аварийно остановится и на экране будет выведена следующая информация:

```
Введите первое число:
2
Введите второе число:
0

Unhandled Exception: System.DivideByZeroException: Attempted to
divide by zero.
   at ExceptionDemo.Main()
```

Для того чтобы корректно обработать ситуацию деления на ноль, можно заменить тип объекта в блоке ***catch*** на *DivideByZeroException*.

```
using System;

class ExceptionDemo {
  public static void Main () {
    try  {
      Console.WriteLine ("Введите первое число:");
      int n = int.Parse (Console.ReadLine());
      Console.WriteLine ("Введите второе число:");
      int m = int.Parse (Console.ReadLine());
      int res = n / m;
      Console.WriteLine ("Res={0}", res);
    }
    catch (DivideByZeroException) {
      Console.WriteLine ("Второе число не может быть нулем");
    }
  }
}
```

Теперь даже если второе число будет равно 0, программа не будет аварийно остановлена, а будет выведено соответствующее сообщение и выполнение программы будет продолжено.

Обработка нескольких исключений

Если при выполнении программного кода в блоке ***try*** могут возникнуть исключения разных типов, которые необходимо по-разному обработать, то необходимо создать дополнительные блоки ***catch***.

```
using System;

class ExceptionDemo {
  public static void Main ()   {
    try {
      Console.WriteLine ("Введите первое число:");
      int n = int.Parse (Console.ReadLine());
      Console.WriteLine ("Введите второе число:");
      int m = int.Parse (Console.ReadLine());
      int res = n / m;
      Console.WriteLine ("Res={0}", res);
    }
    catch (DivideByZeroException)     {
      Console.WriteLine ("Второе число не может быть нулем");
    }
    catch (FormatException) {
      Console.WriteLine ("Неправильный формат целого числа");
    }
  }
}
```

Такой механизм работает следующим образом: в случае, если происходит исключение в блоке ***try***, то исключение последовательно подставляется в каждый блок ***catch*** до тех пор, пока тип исключения не будет соответствовать (приводиться к) типу, указанному в блоке ***catch***. Если ни один блок ***catch*** не подходит, то исключение будет переброшено далее в вызывающий метод (в рассматриваемом примере для метода Main вызывающим является системный метод среды Microsoft .NET Framework).

Если необходимо в блоке catch обработать все возможные исключения, то последний блок catch должен выглядеть следующим образом:

```
catch (Exception e) {
  …
}
```

Генерирование исключений

Для полноценной работы с исключениями недостаточно уметь только их перехватывать и обрабатывать. При разработке своих методов необходимо уметь *генерировать* (*throw – "бросать"*) исключения. Создание и "бросание" исключения вызывающему методу производится с помощью оператора ***throw***. Например:

```
…
if (m == 0) {
  string ex = "Второй аргумент равен нулю.";
  throw new DivideByZeroException(ex);
}
res = n / m;
```

…

С помощью оператора ***throw*** можно генерировать не только исключения, имеющиеся в библиотеке Framework Class Library общеязыковой среды выполнения Common Language Library, но и исключения, которые могут быть созданы дополнительно программистом. Информация о том, каким образом разрабатываются свои исключения, может быть получена в [1].

Если после обработки исключения в блоке ***catch*** необходимо передать его дальше, то необходимо также воспользоваться оператором ***throw***:

```
catch (DivideByZeroException e) {
  …
  throw e;
}
```

Стоит отметить, что из блока ***catch*** можно генерировать исключения любого (а не только обрабатываемого) типа.

Класс System.Exception

Все классы, описывающие исключения, являются потомками класса *System.Exception.* Прямыми потомками этого класса является классы *ApplicationException* и *SystemException*.

Класс *ApplicationException* используется для создания пользовательских исключений.

Класс *SystemException* является базовым практически для всех исключений, определенных в среде Microsoft .NET Framework. Этот класс содержит несколько полезных свойств и методов, позволяющих получить подробную информацию о произошедшем событии. Вот некоторые из них:

- *HelpLink* – хранится ссылка на справочный файл, в котором можно найти дополнительную информацию об исключении,
- *Message* – хранится текстовое сообщение об ошибке,
- *Source* – хранится имя приложения или объекта, которое активировало исключение,
- *TargetSite* – хранит имя метода, передавшего исключение.

Более подробная информация о классе System.Exception и правилах создания пользовательских исключений может быть получена в [1-3].

Литература

1. Рихтер Д. Программирование на платформе Microsoft .NET Framework. – М.: Издательско-торговый дом "Русская Редакция", 2002.

2. Microsoft Developer Network (MSDN) (http://msdn.microsoft.com)
3. Байдачный С.С. .NET Framework. Секреты создания Windows-приложений. – М.: СОЛОН-Пресс, 2004.