第66章 例外の種類を知る

前章のプログラムでは、例外が発生したことはわかってもその種類まではわかりませんでした。この章では、もう少し詳しい例外処理について解説します。

例外はクラスとして表現されます。その大元はExceptionクラスでSystem名前空間で定義されています。

すべての例外クラスはExceptionクラスから派生しています。また、このクラスからApplicationExceptionとSystemExceptionクラスが派生しています。ApplicationExceptionクラスは、ユーザープログラムによって例外がスローされます。スローとは例外を表すオブジェクトがプログラムに送信されることで、俗に「投げる」とも言います。

SystemExceptionクラスは、共通言語ランタイムが生成します。

この2つは、単にシステム定義の例外と、アプリケーション定義の例外を区別するだけの働きしかありません。

まずは、前章を少し発展させたtry-catch構文を見てみましょう。

```
try
{
    //例外が発生するかも知れない処理
}
catch （例外クラス型 変数）
{
 //例外処理
}
```

catch節に例外クラス型とその変数が加わっています。変数は無くてもかまいません。そのかわり、変数を利用した便利な処理ができません。

このように、書くとcatch節で指定した例外以外は捕捉することができません。

ますは、これにException型を指定してみましょう。これなら、すべての例外が捕捉されます。

```
// exception02.cs

using System;

class exception02
{
    public static void Main()
    {
        int x = 100, z;

        for (int i = 10; i > -11; i--)
        {
            try
            {
                z = x / i;
            }
            catch (Exception e)
            {
                Console.WriteLine("Message = {0}", e.Message);
                Console.WriteLine("Source = {0}", e.Source);
                Console.WriteLine("GetType = {0}", e.GetType());
                Console.WriteLine("TargetSite = {0}", e.TargetSite);
                z = 99999;
            }
            Console.WriteLine("{0} / {1} = {2}", x, i, z);
        }
    }
}
```

Exceptionクラスのプロパティには Message,Source,TargetSiteなどがあります。

```
public virtual string Message { get; }
```

これは、現在の例外の説明文を取得します。

```
public virtual string Source { get; set; }
```

例外発生の原因となったアプリケーション、オブジェクトの名前を設定もしくは取得します。

```
public MethodBase TargetSite { get; }
```

現在の例外をスローしたメソッドを取得します。

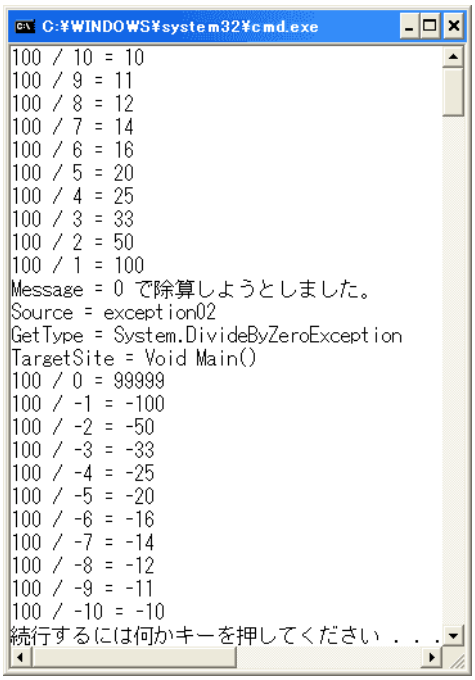
MethodBaseクラスは、メソッドとコンストラクタに関する情報を表します。多数のメンバが 存在します。

Exceptionクラスのメソッドには、GetTypeなどがあります。

```
public override sealed Type GetType ()
```

現在のインスタンスの型を取得します。

では、実行結果を見てみましょう。



実行結果を見ると、投げられた例外はDivideByZeroExceptionで あることがわかります。

主な例外クラスには、次のようなものがあります。

例外クラス	内容
ArgumentException	メソッドに渡された引数に無効なものがある場合にスローされます
ArgumentNullException	nullを引数として受け付けないメソッドにnullを渡したときスローされます
ArgumentOutOfRangeException	パラメータが指定の範囲を超えたときスローされます
DivideByZeroException	整数値または実数値を0で除算しようとした場合スローされます
IndexOutOfRangeException	配列の境界を越えてアクセスしようとしたときにスローされます
InvalidOperationException	無効なメソッド呼び出しが行われた場合にスローされます

catch節を複数設けることも可能です。

```
try
{
 //例外が発生するかもしれない処理
}
catch (例外クラス型1 変数)
{
 //例外処理
}
catch (例外クラス型2 変数)
{
 //例外処理
}
...
```

この場合、例外が発生して最初に捕獲されたcatch節以降は、捕獲されません。

```
// exception03.cs

using System;

class exception03
{
    public static void Main()
    {
        int[] ar = new int[3] { 1, 2, 3 };

        for (int i = 0; i < 5; i++)
        {
            try
            {
```

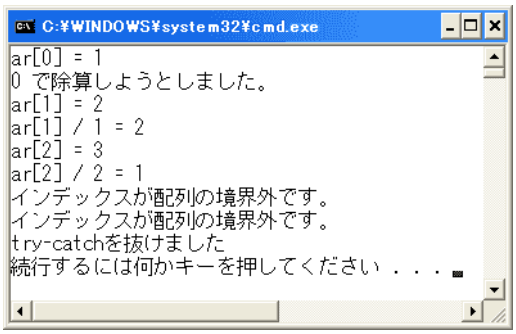
```
                Console.WriteLine("ar[{0}] = {1}", i, ar[i]);
                Console.WriteLine("ar[{0}] / {1} = {2}", i, i, ar[i] / i);
            }
            catch (IndexOutOfRangeException o)
            {
                Console.WriteLine(o.Message);
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }
        }
        Console.WriteLine("try-catchを抜けました");
    }
}
```

try-catch構文がfor文で何回も繰り返されます。

最初にi = 0の時、ar[i] / iの計算で、0での除算例外が発生します。これは、catch (Exception e)で捕獲されます。

次にi = 3とi = 4の時、catch (IndexOutOfRangeException o)で捕獲されます。その後の catch (Exception e)では、捕獲されません。

実行結果は次のようになります。



[C# Index] [総合Index] [Previous Chapter] [Next Chapter]

Update 11/Oct/2006 By Y.Kumei