SKILL UP!

将来にわたって使えるスキルを身につけるために

オブジェクト指向ではじめるプログラミング

日向 俊二
HYUGA, Shunji

第4回

C#の例外オブジェクト

Technology Tools

- ❑ Visual Basic .NET
- ☑ Visual C# .NET
- ❑ SQL Server 2000
- ❑ Oracle 9i
- ❑ Access 2002
- ❑ ASP.NET
- ❑ Internet Information Services
- ☑ Other:
  .NET Framework SDK

Level

★★☆☆☆

Samples

・この記事で取り上げたソースコードおよびサンプルプログラムは、付録CD-ROMの¥DOTNET¥CSHARPディレクトリに収録しています。

¥CUSTOMER
今回作成したサンプル

※この記事では.NET Framework SDKを利用して解説していますが、C#Builder、SharpDevelopでも利用できます（C#コンパイラならすべて利用できます）。

例外ってなんだ？

C#のプログラムでは、実行時に発生する問題に対して、原則として「例外」を使って対処します。

例外は、プログラムの実行中に発生した、プログラムの実行に重大な影響を与える出来事のことです。例外を適切に使うと、プログラム実行中に発生する可能性のある問題に適切に対処することができます。しかし、例外を適切に扱うためには、関数が返すエラーコードを使うような従来のエラー処理とは考え方を変える必要があります。例外も「プログラムの中で扱われるひとつのオブジェクトである」と考えると、複雑な状況も意外に容易に解決できるようになります。

今回は例外と例外処理について、オブジェクト指向プログラミングの観点から考えてみましょう。

C#の例外処理

最初に、C#の基本的な例外処理の方法を整理しておきましょう。

C#の例外処理の構文は、リスト1に示すような形式です。ただし、finallyブロックは必要なければ省略することができます。

catchの行の括弧の最初は「処理する例外クラスの名前」を指定します。

例外クラスは、例外を表わすオブジェクトのクラスです。.NET Frameworkライブラリであらかじめ定義されている例外のうち、とくによく使われる例外クラスを表1に示します。.NET Frameworkで定義されている例外についてのさらに詳しい情報は、MSDNライブラリのドキュメントや、記事末の参考リソースを参照してください。

次に、例外処理の典型的な例をリスト2に示します。この例では、Writeメソッドを使ってプロンプト文字列（入力を促す文字列）を表示し、ReadLineメソッドを使って文字列を受け取りま

リスト1：C#の例外処理の形式

```
try
{
   （例外発生の可能性があるコード）;
}
catch (処理する例外クラス名 変数)
{
   （例外発生時の実行コード）;
}
finally
{
   （必ず実行するコード）;
}
```

す。もし、このときになんらかの例外が発生したら、catchブロックでメッセージを表示します。

この例はいわば“定型”です。ですから、例外が発生する可能性があるメソッドを呼び出すときには、このパターンを理屈ぬきにそのまま、もしくは少し変形させて使えばよいでしょう。例外処理のコードはたいていこんな感じでほとんど同じですから、カンタンです。初心者の方は、“例外処理とはこういうものである”と単純に考えてかまいません。

リスト2：入出力メソッドの例外処理の例

```
using System;

// 例外処理ブロックの始まり
try
{
   // 例外発生の可能性があるコード
   // プロンプトを表示して、
   Console.Out.Write("ID >");
   // 文字列のIDを取得する
   string id = Console.In.ReadLine();
}
catch (Exception e)
{
   // 例外発生時に実行するコード
   // メッセージを表示する
   Console.Out.WriteLine(e.Message);
}
```

例外処理を記述する際に、知っておくと便利なことについて話しておきましょう。

WriteメソッドとReadLineメソッドで発生する可能性がある例外は、次に示す3種類です（.NET Frameworkのドキュメントに明記されています）。

IOException：I/Oエラー
OutOfMemoryException：メモリ不足
ArgumentOutOfRangeException：次の行の文字数がMaxValueを超えた

そこで、厳密に言えば、発生する可能性がある例外ごとに、リスト3のように処理すれば完璧に思えます。しかし、あるコードを実行したために発生する可能性がある例外のうち、発生する可能性がきわめて低い例外まですべてひとつひとつ処理していては時間と労力の無駄です。また、そのような無駄を省くために、.NET Frameworkの例外クラスは階層構造になっています。

ですから、例外をまじめにきちんと処理する方針であっても、比較的発生する可能性が高い例外とそのほかの例外に分け、比較的発生する可能性が高い例外を確実に処理し（たとえば、IOException）、そのほかの例外について

表1：.NET Frameworkライブラリの主な例外クラス

例外クラス	例外がスローされる事態
ApplicationException	致命的ではないアプリケーションエラーが発生したとき
ArgumentException	メソッドに渡された引数のいずれかが無効であるとき
ArithmeticException	算術演算、キャスト演算、変換演算のエラーが発生したとき
DivideByZeroException	整数値または小数値を“0”で除算しようとしたとき
Exception	アプリケーションの実行中に発生する例外の基本クラス
ExecutionEngineException	共通言語ランタイムの実行エンジンに内部エラーが発生したとき
FormatException	引数リストが呼び出されたメソッドの引数リストと一致していないとき
IndexOutOfRangeException	配列の境界外のインデックスで配列の要素にアクセスしようとしたとき
InternalBufferOverflowException	内部バッファがオーバーフローしたとき
IOException	I/Oエラーが発生したとき
NotFiniteNumberException	浮動小数点値が“正の無限大”“負の無限大”“NaN”のいずれかであるとき
OutOfMemoryException	プログラムの実行のためのメモリが不足しているとき
OverflowException	checkedコンテキストで、算術演算、キャスト演算、変換演算の結果オーバーフローが発生したとき
StackOverflowException	スタックがオーバーフローしたとき