けん太のプログラミング修行記

第13回「けん太、例外処理問題を処理する」の巻 ―その2―

構造化例外処理について考える

碗仔 けん太（Pochi Company）
WANKO, Kenta

Technology Tools

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

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\KENTAディレクトリに収録しています。

\EDITORVB
VB.NET版のテキストエディタプログラム

\EDITORCS
C#版のテキストエディタプログラム

\DEC2HEX
本誌4月号の10進数16進数変換プログラム（再収録）

Visual Basic 6.0以前では、実行中に発生したプログラムのエラー処理を、基本的にはOn Errorステートメントを使って行なっていました。Visual Basic .NETになって、実行中に発生したプログラムのエラーは構造化例外処理機構を使って対処するようになりました。そこで、Try…Catch…Finally（Visual C# .NETならtry…catch…finally）を使う構造化例外処理を前回のプログラムで試してみました。構造化例外処理という言葉からは、なにか難しそうな感じを受けましたが、プログラムを行なってみるとコードの書き方にとくに難しいことはないということがわかりました。

しかし、Visual Basic 6.0以前のOn Errorステートメントを使った非構造化例外処理からVisual Basic .NET（以下VB.NET）では構造化例外処理に変わったことや、Visual C# .NET（以下C#）でも構造化例外処理が採用されているということなどから、構造化例外処理にはこれまでのOn Errorを使う方法にはないメリットが何かあるようです。そこで、今回は非構造化例外処理と構造化例外処理の、見た目（構文）の違いについてだけでなく、その特性について詳しく調べてみましょう。

構造化例外処理の基本の基本

VB.NETやC#の例外処理の構文は本誌でもすでに何度も登場していますが、念のために最初に簡単におさらいしておきましょう。

VB.NETではTry…Catch…Finallyステートメントを使って、C#ではtry {…} catch {…} finally {…}の形式を使って、エラーが発生する可能性があるコードと発生したエラーに対処するためのコードを書けばよいのでしたね（表1）。

そして、前回はVB.NETのプログラムで例外処理をリスト1のように行ないました（リスト1のコードは付録CD-ROMに収録してあるサンプルプログラムDec2Hexに記述してあります）。

これでプログラムはフツーに動作したので、ソースコードには問題がなさ

そうです。しかし、我が社（ポチ・カンパニー）[注1]のボスであるポチさんによると、これでは完璧とはいえないそうです。その理由は、例外をオブジェクトとしてとらえていないからだそうです。

そうなんです。実は、例外はオブジェクトなのです。しかし、表1やリスト1のコードを見ていても、例外がオブジェクトであるという感じがしません。

もう少し詳しい構造化例外処理の基本

Visual Studio .NETのドキュメント（MSDNライブラリ）をもう少し詳しく調べてみましょう。すると、VB.NETのTry…Catch…Finallyステートメントには次のような使い方もあるということがわかりました（リスト2）。

プログラミング言語がC#であるなら、これはリスト3のようになるようです。

これらは、発生した例外をすべて処理するのではなく、［例外クラス］で指定した例外だけを処理します。

指定した例外だけを処理するということはどういう意味でしょう？

選択して処理できるということは、例外にはいろいろな種類があるということに他なりません。そして、それぞれの種類ごとに例外クラスというクラスが定義されていることに違いありません。

そこで、例外クラスを調べてみると、ありましたアリマシタ。ざっと調べてみただけでも、記事末の表Aに示すような種類の例外（例外クラス）があり

表1：例外処理の形式

VB.NET	C#

Try
（例外発生の可能性があるコード）
Catch
（例外発生時の実行コード）
Finally
（必ず実行するコード）
End Try
	try
	{
	（例外発生の可能性があるコード）;
	}
	catch
	{
	（例外発生時の実行コード）;
	}
	finally
	{
	（必ず実行するコード）;
	}

リスト1：Dec2Hexプログラムの例外処理コード

```
' 16進のテキストボックスの表示を更新する
Private Sub UpdateHex()
    Dim s, ss, txt As String
    Dim v As Integer
    s = txtDec.Text.Trim()
    Try
        ss = StrConv(s, VbStrConv.Narrow)
        v = Integer.Parse(ss)
        txt = v.ToString("X")
    Catch
        MessageBox.Show("10進数の値が不正です。" + ControlChars.Cr + s)
        txt = ""
    Finally
        txtHex.Text = txt
    End Try
End Sub
```

リスト2：キーワードAsを使うVB.NETの例外処理

```
Try
    （例外発生の可能性があるコード）
Catch ＜変数＞ As ［例外クラス］
    （指定した例外クラスの例外発生時の実行コード）
Finally
    （必ず実行するコード）
End Try
```

※）本稿では、リスト2やリスト3のコード行の中で「例外クラス」や「変数」を目で見てわかりやすくするために［例外クラス］や［変数］と表記しています。[ ]や＜＞が省略可能であることやオプションであることを示しているワケではありません。

リスト3：例外クラスを指定して処理するC#の例外処理

```
try
{
    （例外発生の可能性があるコード）
}
catch （［例外クラス］ ＜変数＞）
{
    （指定した例外クラスの例外発生時の実行コード）
}
finally
{
    （必ず実行するコード）
}
```

注1）本稿にでてくる会社や登場人物はフィクションであり、実在する製品、会社、人物、犬種、そのほかモロモロあれこれとはいっさい関係がありません。念のため。