構造化例外プログラミング再入門

>>> 例外処理を確実に身につけよう

初音 玲 HATSUNE, Akira

CHAPTER 2

はじめに

Visual Basic 6.0までのエラー処理は「On Error Goto」を使用していたため、構造化プログラミングで統一するためにGoto文を排除しようと思っても、「ただしエラー処理のOn Error Goto文は除く」の一文を加えるしかなかった。

しかし、Visual Basic .NET 2002以降では、Try～Catch～End Tryが追加され、やっとエラー処理も構造化例外処理として記述できるようになった。だが、単純にOn Error Gotoを記述するエラー処理のころに比べると例外処理はきちんと設計する必要が生じて、面倒だからと例外処理を入れないようなプログラムコードに出会うことも多くなってきた。

本稿では、Try～Catch～End Try（Try制御構造）による構造化例外の基礎からもう一度見直して正しい構造化例外のありようを検証したいと思う。

最初の一歩

On Error Goto文に代わる例外処理の基本は図1のようになる（サンプル：SimpleCatch）。

Tryからが構造化例外処理になっていて、Try～Catchの直前（Tryブロック）で例外が発生したら、Catch句以降の行に制御が移り、End Tryまで（Catchブロック）の間に書かれた行を実行する。

Catch句に書かれているException変数「ex」にはさまざまな例外情報が設定されている。当然、例外発生時の処理として代表的なものである例外メッセージ出力のためのメッセージもMessageプロパティから取得できる。その他にも、StackSourceプロパティにはリスト1のような情報が設定される。この情報は例外が発生したときのメソッド呼び出しが下から上に記録されているので例外発生箇所の情報取得に役立つ。

なお、IDE上で実行した場合やPDBファイルがある場合は例外が発生したソースコード行の位置まで特定してくれるが、リスト1は、より実践的にするためにPDBファイルを配置せずにアセンブリ（EXEファイル）を実行したときの結果になっている。

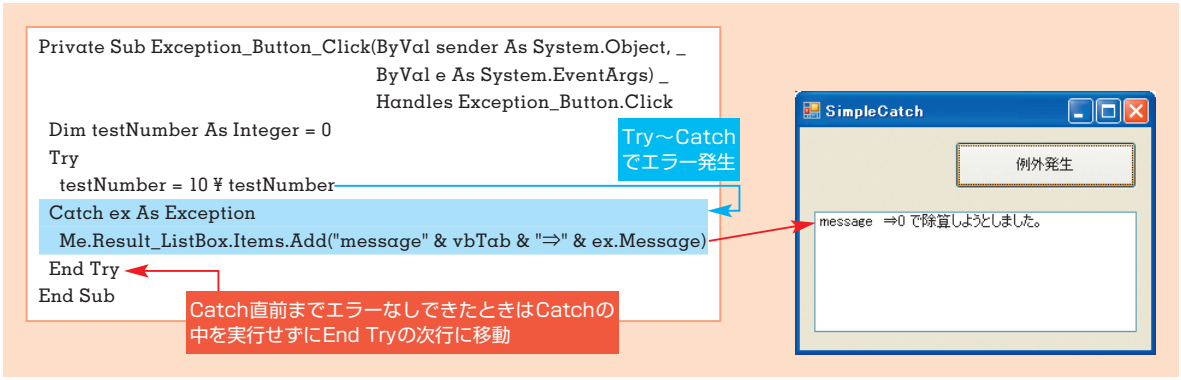
レベル >>> Level
1 2 3 4 5

言語 >>> Language
- Visual Basic

ツール >>> Tool
- Visual Studio 2005 Professional
- Visual Studio .NET 2003 Professional

サンプル >>> Sample
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、
http://www.shoeisha.com/mag/windev/
からダウンロード可能です。

図1：構造化例外の基本的な動き



リスト1：サンプル実行結果（PDBファイルなし）

```
場所 System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)
場所 System.IO.FileStream.Init(String path, FileMode mode, FileAccess access, Int32 rights, ……
場所 System.IO.FileStream..ctor(String path, FileMode mode, FileAccess access, FileShare sha……
場所 System.IO.StreamReader..ctor(String path, Encoding encoding, Boolean detectEncodingFrom……
場所 System.IO.StreamReader..ctor(String path)
場所 Hatsune.Sample.Windev0612.NestCatch2.nestLevel2()
場所 Hatsune.Sample.Windev0612.NestCatch2.nestLevel1()
場所 Hatsune.Sample.Windev0612.NestCatch2.Exception_Button_Click(Object sender, EventArgs e)
```

Finally句とは

Try制御構造ではCatch句の後ろにFinally句も指定できる（図2・サンプル：CatchFinally）。

この場合、例外が発生したときのみ実行されるCatchブロックはFinally句の直前行までになる。一方、Finally句の次の行からEnd TryまでのFinallyブロックは、例外発生の有無に関わらず、Try制御構造から抜けるときに必ず実行するコードを記述する。

たとえば、Tryブロックの先頭でマウスカーソルを砂時計にしているような場合は、デフォルトに戻すコードはEnd Tryの後に記述するのではなく、Finallyブロックで記述する。もちろん、Tryの前で砂時計にしているのならばEnd Tryの後でデフォルトに戻すように記述する。

このように構造化例外ではTry制御構造の構造化例外処理の範囲を意識することで後処理の抜けなどが防止できる利点がある。

図2：Finally