けん太のプログラミング修行記

第14回「けん太、例外処理問題を処理する」の巻 ―その3―

例外のスローと例外クラスの定義について

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。

\TELCTRLVB
電話番号コンポーネントとテストプログラム（VB.NET版）

\TESTVB
TELCTRLVBのテストプログラムのソース

\TELCTRLCS
電話番号コンポーネントとテストプログラム（C#版）

\TESTCS
TELCTRLCSのテストプログラムのソース

.NETになって、構造化例外処理という強力なエラーリカバリー機能がVisual Basic .NET（以下VB.NET）やVisual C# .NET（以下C#）に導入されました。その結果、特定のエラーを例外というかたちで処理することができるようになりました。しかも、例外クラスが階層構造を形成していることを利用すると、一連の例外をまとめて処理することもできるということが、前回はわかりました。また、例外はクラスのインスタンスであるということ、つまり例外もオブジェクトであるということも確認しました。
今回は例外をスローするということと、独自の例外クラスを定義することについて考えてみましょう。

例外のスロー

.NET Frameworkライブラリであらかじめ定義されている異常事態[注1]がプログラムの中で発生した場合、システム例外としてプログラムに自動的に通知されます。たとえば、値を0（ゼロ）で除算しようとしたときにはDivideByZeroExceptionという例外、プログラム実行のためのメモリが不足しているときにはOutOfMemoryExceptionという例外がプログラムに自動的に通知されます。

このように、プログラムに例外を通知することを、「例外をスロー」するというのでしたね。

プログラムで異常事態が発生したときに例外は自動的にスローされますが、そのほかに、プログラマが自分で例外をスローすることもできるそうです。

一般的に、例外をスローする主な状況には、次の2種類があるようです。

・プログラムで異常事態の発生を知らせるために例外をスローする
・例外処理の構文の中で例外をまったく処理しないで同じ例外を再スロー

注1）.NET Frameworkクラスライブラリであらかじめ定義されていて自動的にスローされる主な例外については、先月号の本稿（P.133）に掲載している「表A：.NET Frameworkライブラリの主な例外」を参照してください。

するか、問題の一部を処理しないで別の例外をスローする

プログラムで異常事態の発生を知らせるための例外は、プログラムの中で次のコード[注2]を実行するだけでスローすることができます。

```
' VB.NETの場合
Throw New ［例外クラス］

// C#の場合
throw(new ［例外クラス］("例外の説明文字列"));
```

したがって、その時点でスローするべき適切な［例外クラス］の名前さえわかれば、コード自体は簡単です。既存の適切な［例外クラス］がない場合には自分で例外クラスを定義すればよいのですが、このことについてはあとでもう1度検討しましょう。

例外処理における例外のスロー

スローされた例外を処理するためのプログラム部分である「例外処理」で例外をスローするというのは奇妙に感じるかもしれません。しかし、実際のプログラミングの場面では例外処理の中で例外をスローしたり、例外処理の中で現在処理している例外を再びスローすることはよくあるようです。このような状況としてよく見かけるコードは、リスト1（VB.NET）やリスト2（C#）のかたちであるのが普通ですね。

例外処理の危険な罠

通常のプログラムコードの中で発生した異常事態を単に知らせるために例外をスローするにしても、例外処理の中で例外をスローするにしても、例外をスローするには、単にプログラムの中でThrowまたはthrowを使えばよいので、難しいことはなにもなさそうです。しかし、例外処理にはとても危険な罠が待ち構えています。それは、処理していない例外があると、実行時にプログラムがメッセージを表示して実行中のプログラムが停止してしまうということです。

みなさんも、実際に自分でプログラミングしているなら、図1や図2のダイアログボックスを目にしたことがあるはずです（これらのダイアログボックスについては本誌2003年4月号の本稿も参照してください）。

これはハンドルされていない例外、つまり処理していない例外があることの通知です。しかし、プログラムの実行中にこのダイアログボックスはあまり見たくないですし、ユーザーがプログラムを実行しているときにこの種のダイアログボックスが表示されることによりクレームが来るような事態も避けたいものです。

そのためには、「例外は処理しなければならない」という単純なルールを忘れないようにすればよいわけですが、自分でThrowまたはthrowを使って例外をスローするときに、この点についてどのように考えればよいでしょうか。

リスト1：VB.NETの例外処理における例外のスロー

```
Try
  （例外発生の可能性があるコード）
Catch ＜変数＞ As［例外クラス1］
  （指定した例外クラスの例外発生時の実行コード）
  Throw New［例外クラス］' 例外をスローする
Catch ＜変数＞ As［例外クラス2］
  （指定した例外クラスの例外発生時の実行コード）
  Throw ＜変数＞          ' 同じ例外を再スローする
Finally
  （必ず実行するコード）
End Try
```

リスト2：C#の例外処理における例外のスロー

```
try
{
  （例外発生の可能性があるコード）
}
catch (［例外クラス1］ ＜変数＞)
{
  （指定した例外クラスの例外発生時の実行コード）
  throw(new ［例外クラス］());  // 例外をスローする
}
catch (［例外クラス2］ ＜変数＞)
{
  （指定した例外クラスの例外発生時の実行コード）
  throw;                // 同じ例外を再スローする
}
finally
{
  （必ず実行するコード）
}
```

注2）本稿では、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＜ ＞が省略可能やオプションであることを示しているわけではありません。