Exceptionのしくみ
～いまさら聞けない例外設計～

by　mxb

agenda

- 例外って？
- 例外は何が便利？
- VB6時代の例外処理は？
- .NETでの例外処理は？
- .NETでOnErrorは使えるの？
- Exceptionの仕組み
- System.Exceptionと派生クラス
- Application Exceptionとは？
- なぜ例外設計が必要なのか？
- 例外設計のコツ

例外って？

- 例外とは…
  - 例外（れいがい）とは、通例の法則や規則、規定から外れること、あるいは外れたものをいう。
  - プログラミングでは、プログラムがある処理を実行している途中で、なんらかの異常が発生することを例外という。

出典: フリー百科事典『ウィキペディア（Wikipedia）』

- つまり…
  - ロジックとして正常処理以外の動作は全て例外
- 例外は大きく分けて…
  - OSやハード、ネットワークなどのシステムの基盤的な部分から発生する例外と、システムの機能要件(DBにデータがあるはずなのに無い等)による例外がある

例外は何が便利？

- 例外がないと…
  - ロジックで発生しうるエラーに対して抜けなく全て設計し、エラーの発生原因を解析するための情報を収集し、共通なログを出力し、共通なメッセージを出力し…
  - さらに、処理を中断させるか、リカバリーさせるか、無視して続行させるか…
  - これら全てを設計者は設計し、プログラマは実装し、テスターはテストし…

⇒はっきり言って、無理です

例外は何が便利？

- 例外を使うと…
  - 言語のランタイムや共通関数ライブラリが提供している例外を使用すると、システムの基盤的な部分のエラーは、ほとんど例外として捉える事が出来る
  - エラーの発生原因を解析するための情報の収集が比較的に容易になる

⇒つまり、設計者もプログラマもテスタも少し楽が出来る

VB6時代の例外処理は？

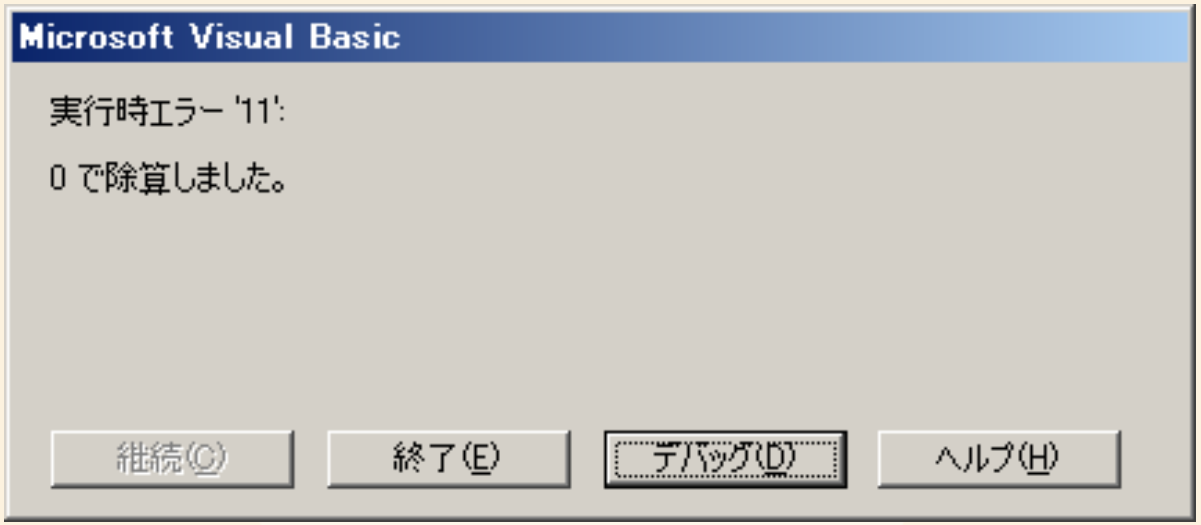
- Visual Basic 6.0では「OnError」を使用してエラーをハンドルする

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        On Error GoTo errorHandler
        Dim i As Integer
        TextBox1.Text = " "
        For i = -2 To 2
            Dim j As Integer
            j = 10 ¥ i
            TextBox1.Text = TextBox1.Text + j.ToString() + " "
        Next
        On Error GoTo 0
        Exit Sub
errorHandler:
        If Err.Number = 11 Then
            TextBox1.Text = TextBox1.Text + "∞" + " "
            Resume Next
        End If
        Error Err.Number
    End Sub
```

VB6時代の例外処理は？

- Visual Basic 6.0のエラー処理

「On Error」がないと. . .



.NETでの例外処理は？

- Microsoft .NETでは「**Try...Catch...Finally ブロック**」でエラーをハンドルする

```
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button2.Click
      Dim i As Integer
      TextBox1.Text = " "
      For i = -2 To 2
        Dim j As Integer
        Try
          j = 10 ¥ i
        Catch ex As SystemException
          TextBox1.Text = TextBox1.Text + "∞" + " "
        End Try
        TextBox1.Text = TextBox1.Text + j.ToString() + " "
      Next
  End Sub
```

.NETでの例外処理は？

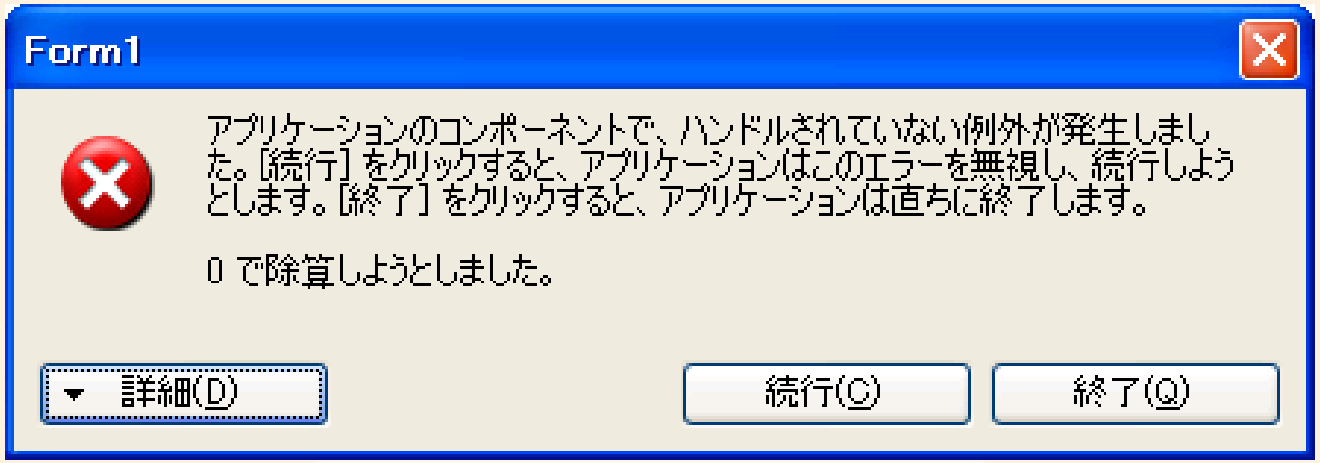
- Microsoft .NETでは「**Try...Catch...Finally ブロック**」でエラーをハンドルする

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button3.Click
      Dim i As Integer
      TextBox1.Text = " "
      For i = -2 To 2
        Dim j As Integer
        Try
          j = 10 ¥ i
        Catch ex As DivideByZeroException
          TextBox1.Text = TextBox1.Text + "∞" + " "
        End Try
        TextBox1.Text = TextBox1.Text + j.ToString() + " "
      Next
  End Sub
```

.NETでの例外処理は？

- Micrisoft .NETのエラー処理

「Try ～Catch～End Try」がないと. . .



.NETでOnErrorは使えるの？

- 使えます

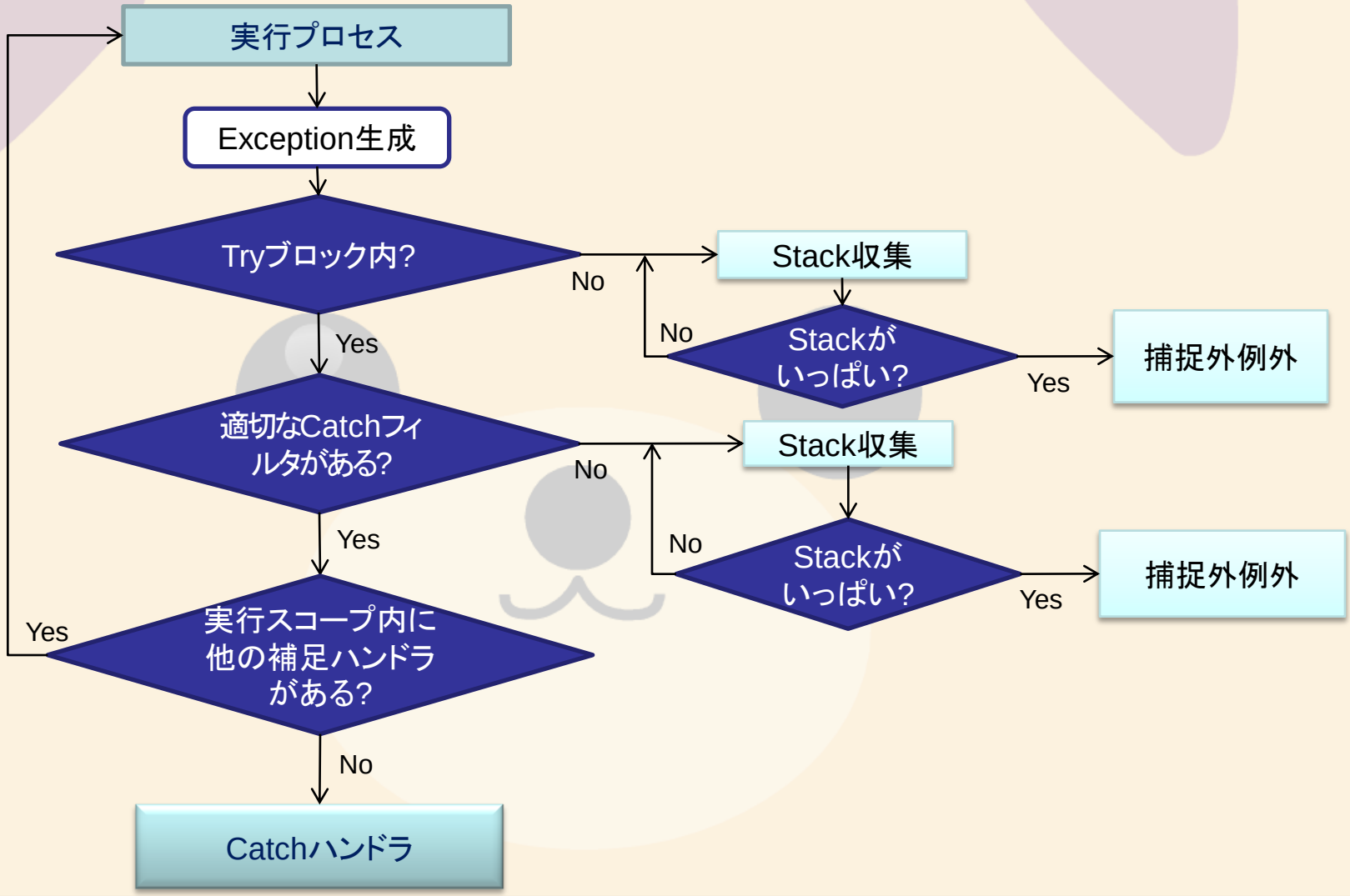
Protected Sub OnError(ByVal e As ErrorEventArgs)

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
    On Error GoTo errorHandler
    Dim i As Integer
    TextBox1.Text = " “
    For i = -2 To 2
        Dim j As Integer
        j = 10 ¥ I
        TextBox1.Text = TextBox1.Text + j.ToString() + " “
    Next
    On Error GoTo 0
Exit Sub
errorHandler:
    If Err.Number = 11 Then
        TextBox1.Text = TextBox1.Text + "∞" + " “
        Resume Next
    End If
    Error Err.Number
End Sub
```

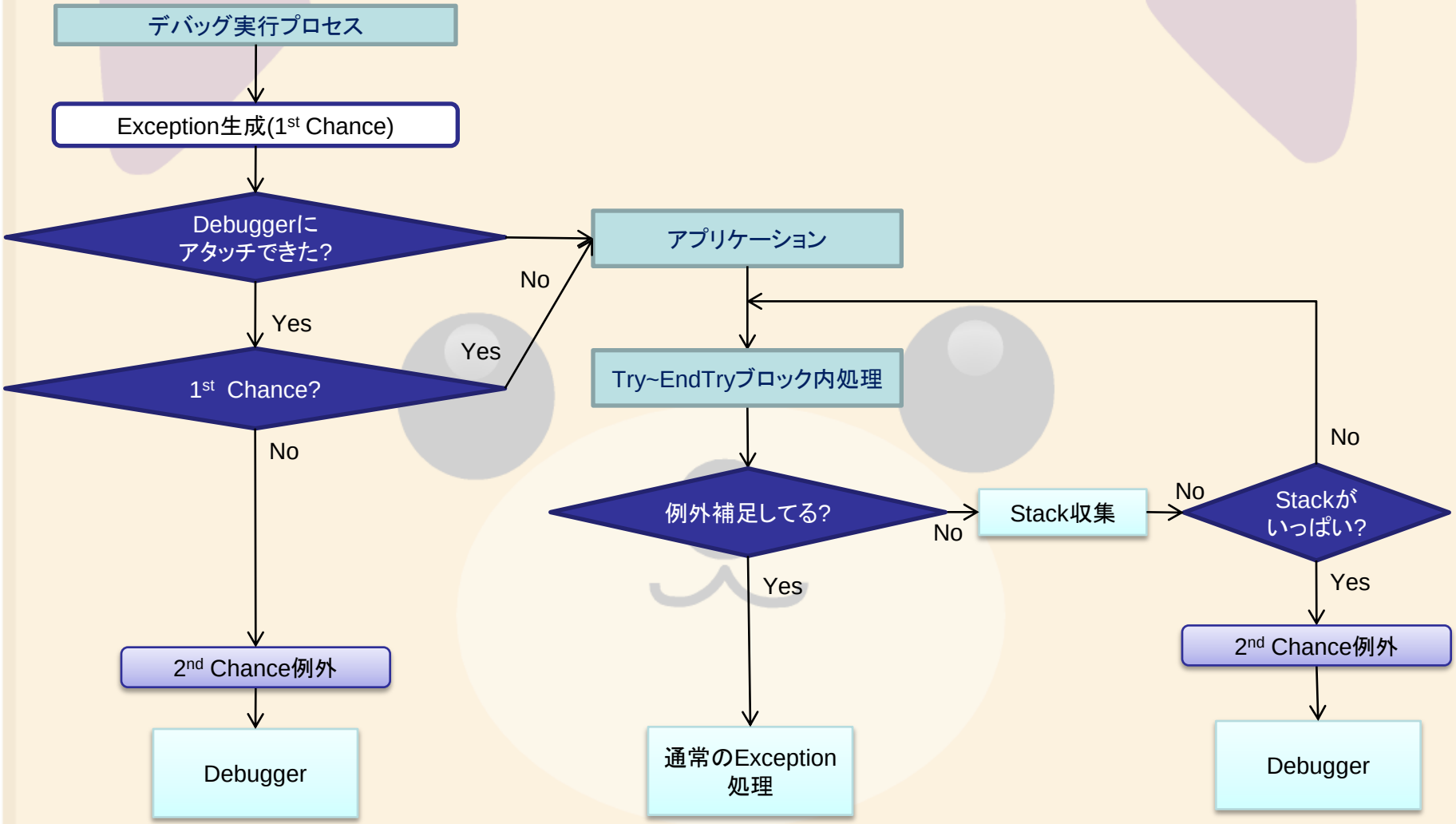
.NETでOnErrorは使えるの？

- しかし、使わないほうがいい
  - なぜなら…

Exceptionの仕組み



Exceptionの仕組み



Exceptionの仕組み

- MSIL アセンブラでMSILコードをみてみよう
  - MSIL アセンブラ (Ilasm.exe) でMicrosoft Intermediate Language (MSIL) コードを含む、ポータブル実行可能 (PE) ファイルを使用して、Ilasm.exe に対する入力として適したテキスト ファイルを作成します。

出典: MSIL 逆アセンブラ (Ildasm.exe)
http://msdn2.microsoft.com/ja-jp/library/f7dy01k1(VS.80).aspx

System.Exceptionと派生クラス

- MSDN Onlimeから拾ってきました
  - System.SystemExceptionの1段目の派生クラスは123種類
  - System.SystemExceptionのすべての派生クラスは225種類
  - これらすべての派生クラスによって捉えられる例外も違うし、使用するメモリ、処理されるCPUサイクルも違う

⇒そのロジックに最適なExceptionクラスを使用する

Application Exceptionとは？

- 一般的に、システムの基盤的な部分から発生する例外ではなく、システムの機能要件からくる例外
- システムの基盤的な部分から発生する例外であるが、システムの機能要件からシステム例外ではなく、アプリケーション例外として扱う場合

⇒これら例外を捉えるためにユーザが定義する例外クラス
Application Exceptionクラスを継承したクラスとして作成する

なぜ例外設計が必要なのか？

- 例外設計をしないと…
  - さまざまなレベルの技術を持ったプログラマが独自に例外を捉える仕組みを実装することになる。
- システム内で複数の例外を捉える仕組みがあると…
  - 冗長コードが出来てしまう。
  - プログラマによって取得できる例外情報が異なってしまう。
  - 例外情報の出力漏れが発生する可能性がある。
  - 例外処理の均一的な品質を確保できない。

例外設計のコツ

- システム内で発生する様々な例外の情報を統一的なインタフェースで取得できるようにする。
- 例外はできるだけ、発生した箇所に近い所で捉える。
- 例外情報の出力はできるだけ、1箇所で出力する。
- 出力する例外情報は解析しやすい状態で出力する。

例外設計のコツ

- しちゃいけないこと
  - 不必要なTry…Catchは書かない!
    - 振舞い(動作)はデフォルトで正常であることを前提に!
  - 再スローには “throw e” を使用する
    - 例外情報(スタックトレースなど)が消えてしまう!
  - より大きな(上位の)例外でラッピングする
    - 設計(コーディング)したあなた以外エラーの詳細な中身は知りたくない!
  - システムの共通パスで例外を捉えるようにする

例外設計のコツ

- 最小限の例外処理で設計する
  - Exceptionはエラーの時のみに使用する
    - Exceptionをイベントとして使用しない
    - Exceptionはシステム資産(メモリ、CPU)を多く使う
  - OnErrorを使用せずにTry…Catchを使用する
    - エラー制御の流れの設計をしっかりすることがより良いコードを生み出す
    - 自動生成コードを使用する(Bestじゃないけどね)
      - 再利用性が高くなる
    - エラーオブジェクトをスローしない

参考出典

- MSDN Online
  - MSIL 逆アセンブラ (Ildasm.exe)

    http://msdn2.microsoft.com/ja-jp/library/f7dy01k1(VS.80).aspx
  - Exception クラス

    http://msdn2.microsoft.com/ja-jp/library/system.exception(VS.80).aspx
- Exception Management in .NET
  - (Patterns & practices)