快速 脱!休日出勤 デバッグ技法

.NETアプリケーション障害解析の極意

第6回

コーディングのミスを防ぐ方法 ―プログラミング工程―

株式会社NTTデータ
飯山 教史
IIYAMA, Takashi

.NETで ミスが多いコードとは?

私は.NETプログラムのソースコードをレビューする機会が多い。レビューでは日本コンピュウェアのDevPartner Studio [注1] の静的ソースコード解析機能を利用している。このツールはさまざまなルールを定義しており、それをもとに開発者の書いたコードをレビューしている。

これまでさまざまなプロジェクトでレビューを行なってきたが、このツールによってよく指摘されるルール、つまり開発者が多くミスをするポイントは以下の3点に絞られることがわかった。

- アクセス修飾子の記述
- 例外処理の記述
- ガベージコレクションの記述

これはプログラマがこれらの知識をよく把握せずに開発していることを意味している。言うまでもなくこれらは安定したコードを書くために必要な知識であり、この部分がうまく書けていないということは不安定なコードが数多く展開されていることを意味している。そこで今回は.NETで開発する開発者向けに、上記の3要素に関するノウハウを紹介する。

今回の記事で皆さんがより堅牢性の高いコードを書くためのサポートができたら幸いである。

注1）この製品の詳細情報は以下のURLを参照。
http://www.compuware.co.jp/products/devpartner_fm/

レベル >>> Level
1 2 3 4 5

ツール >>> Tool
- Visual Studio 2005 Team Edition for Software Testers

言語 >>> Language
- Visual Basic

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。

アクセス修飾子

.NETでは、アセンブリが実装情報を保護するカプセル化の境界になる。これはアセンブリが、「実装したクラスやクラス内のフィールド／メソッド」がどこからアクセスできるかを示すアクセス境界のひとつであることを意味している。たとえばフィールドをPublicで宣言すれば、すべてのコードからアクセスできるようになる。また、Privateで宣言すればそのフィールドが宣言されているクラスのメソッドとコンストラクタからしかアクセスできなくなる。このようなアクセスレベルを図にまとめたものが図1である。

開発者は、クラス、およびクラスのメソッドやフィールドを実装する前に

図1のようなアクセスパターンがあることを考慮してアクセス修飾子をコード内の全要素に対して定義しなくてはならない。Visual Basic 2005でのアクセス修飾子は表1のようになっている。アクセス修飾子を何も付けない場合、

・メソッドはPublic
・Dimで宣言されたフィールドは
　Private

になる。

レビューをしていて特に多かったのは、「メソッドにアクセス修飾子を何も付けず、メソッドがPublicになっている」という間違いである。これはセキュリティ上からとても危険な状態なので、設計時にはクラス内の全メンバに対して必ずアクセス修飾子を定義することを忘れないでほしい。

図1：アクセス境界とアクセスパターン

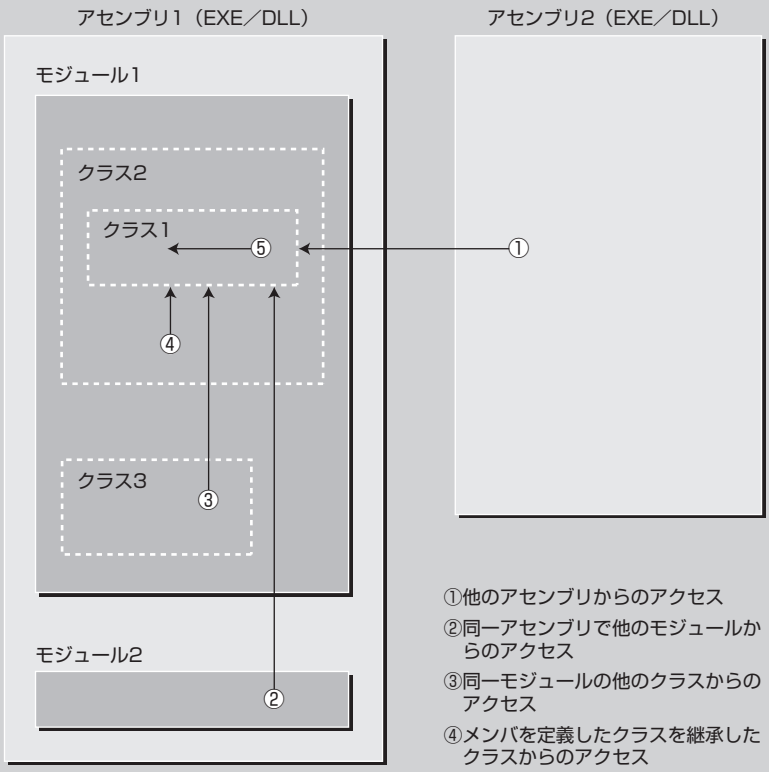

表1：Visual Basic 2005のアクセス修飾子

単位	アクセス修飾子	意味
クラス	Public	アクセス境界なし
	Friend	アセンブリ内部からのみアクセス可能
メンバ[*1]	Public	アクセス境界なし
	Friend	アセンブリ内部からのみアクセス可能
	Protected	宣言されたクラスとそのクラスを継承したクラスからのみアクセス可能
	Protected Friend	宣言されたクラスとそのクラスを継承したクラスからのみアクセス可能。また、アセンブリ内部からもアクセス可能
	Private	宣言されたクラス内部からのみアクセス可能

＊1）クラス内の全要素の総称。

例外処理

Microsoftは.NETで例外処理を利用するようになった。例外処理の最大のメリットは「エラーの発生を無視できない」ことである。例外が発生した後に適切な後処理を行なわないと、CLRはアプリケーションを強制終了する。そのため開発者は必ず例外処理の後処理を記述するようになる。その結果、以前のようにシステムエラーコードを利用しエラー発生を無視して処理を進めるコードが記述できなくなったため、適切なエラー処理が記述されるようになる。

このように、例外処理を利用すればアプリケーションの品質は向上するが、例外処理に慣れている開発者は案外少ない。例外処理に不慣れな開発者は、未処理の例外を多く残し、トラブルを抱えたコードを記述してしまうことがある。

そこで例外処理を記述するための構成要素をおさえ、それぞれの機能を簡単に説明する。例外処理は「Try」「Catch」「Finally」の3種類のブロックから構成される。各ブロックの記述スタイルは以下のとおりである。