開発者必読！

“もしも”のときのデバッグ技法

.NETアプリケーション障害解析

株式会社NTTデータ
飯山 教史
IIYAMA, Takashi

第7回 例外処理の実装のポイント

トラブルを未然に防ぐ

これまで本連載では、トラブルの原因をどのように解析すればよいのかについて説明してきた。その中で何度も解説してきた、障害時のメモリの内容を調査するメモリダンプの解析は「トラブル発生後」の話である。トラブル発生後に障害の原因を解析するトラブルシューティングは、もちろん現場で求められるスキルである。しかしよく知られているように、バグはより早い段階で検出／修正したほうが修正コストが低く抑えられるので、発生するトラブルの数を減らすよう開発段階から努力すべきなのは言うまでもない。

そこで今回から数回にわたって「トラブルを起こさないコードの書き方」について説明していきたい。今回は「例外処理の記述方法」を詳しく説明する。例外処理はその用法を誤るとアプリケーションをダウンさせてしまう、記述が非常に難しいコードである。この記事で適切な例外処理の記述方法を理解し、皆さんのアプリケーションの品質を向上してもらいたい。

Level

1 2 3 4 5

Technology Tools

- ◯ Visual Basic
- ✓ Visual C#
- ◯ Visual C++
- ◯ SQL Server
- ◯ Oracle
- ◯ Access
- ◯ ASP.NET
- ◯ Other:

例外処理とは？

Microsoftは、.NETから全面的に例外処理を利用するようになった。.NET以前はWin32 APIからシステムエラーコードを戻していた。呼び出し側ではそのエラーコードの種類を判別し、障害発生後に必要となる後処理を記述していたのである。コードの状態を通知する手段としてはそれなりに利用できたこの方法も、“解析”という観点で考えると問題があった。たとえば、エラーコードでは発生したエラー以外の情報を伝えられないので、どのコードでその問題が発生したのかがわからなかった。また、エラーコードを呼び出し側に戻すだけで、アプリケーションの処理自体は止まらなかったため、呼び出し側でエラーの発生を無視できるようになっていた。その結果、アプリケーションは予測できない障害を引き起す可能性を持ったまま処理を継続できたのである。エラーが発生したときには、どこでそのエラーが発生したかの情報がまったくないため、解析がとても困難であった。

しかし、例外処理を利用すれば上記のシステムエラーコードによる不具合が解消されるのである。たとえば、あるパラメータがnullであったために例外が発生する場合、例外オブジェクトの

Messageプロパティにエラー原因となったパラメータ名を持たせれば、呼び出し側でその原因がすぐに特定できる。また、例外オブジェクトはスタック情報を記録しているため、どのメソッドの何行目でこの例外が発生しているのかが簡単にわかる。

これらのメリットも含め、例外処理の最大のメリットは「無視できないインパクトを持つ」ことである。コードで後処理をしていない場合、例外が発生したときに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考慮するようになる。これまでシステムエラーコードを利用していたときのような、安易にエラーを無視したコードを記述できなくなるのである。

.NETで提供される例外処理を適切に利用すればアプリケーションの品質は向上するが、Windowsプログラマは例外処理に慣れていないため、.NETに移行した際に潜在的に多くのトラブルを抱えたコードを記述することが多いようである。次の節では例外処理を記述するための構成要素をおさえ、それぞれの機能を説明する。

例外処理のブロック

例外処理は「try」「catch」「finally」の3種類のブロックから構成される。各ブロックの役割は以下のとおりである。

try

このブロックには通常の処理を行なうコードのうち、例外が発生する[注1]可能性のあるコードを記述する。

catch

このブロックには特定の例外に対応したコードを記述する。ブロックで対応する例外は「例外フィルタ」と呼ばれる、catchキーワードの次に記述されている()内で指定される。たとえば、

```
catch(ArgumentNullException ex)
```

と記述すると、ArgumentNullExceptionが発生したときにこのブロック内のコードが実行される。ただし、これ以外の例外が発生した場合には「catch (ArgumentNullException ex)」ブロックのコードは実行されない。

catchブロックには、tryブロックで開始された処理が途中であれば開始前の状態に戻すなど、例外から復旧するために必要なコードを記述する。

finally

このブロックはtryブロックで例外が発生した／しないに関わらず実行される。そのため、確保されたリソースを解放する処理を記述する。

以上が各ブロックの概要である。簡単なサンプルをリスト1に示す。このサンプルを見れば例外処理ブロックがそれぞれどのような処理を行ない、どのような役割をしているか、おおよそ理解できるだろう。

各ブロックの注意点

続いて、例外処理の各ブロックの記述作法をまとめ、それぞれのブロックを記述する際に気をつけてほしい点を説明することにする。

▶tryブロック

catch／finallyブロックを超えるgoto文を記述しない

goto文のジャンプ先がcatch／finallyブロックを越えてしまうと、これらのブロック内のコードが実行されなくなってしまう。このようなコードを記述すると、たとえばfinallyブロックでファイルを閉じるコードを記述していてもファイルが閉じなくなる。

▶catchブロック

例外フィルタでExceptionオブジェクトを指定するときには注意する

.NET Frameworkクラスライブラリには多数の例外型が定義されている[注2]。Exceptionは.NET Frameworkクラスライブラリ準拠の例外オブジェクトの最上位のオブジェクトなので、例外フィルタでExceptionを指定すると.NET Frameworkクラスライブラリ準拠の例外をすべて補足してしまう。そのため、発生していた例外の型に関わらずExceptionを指定したブロックの処理が実行されるようになる。

すべての例外の後処理を丸めてしま

注1）MSDNライブラリにはメソッドから発行される例外が記載されているので、開発時には必ず確認すること。
注2）.NET Frameworkクラスライブラリが提供している例外クラスに関しては以下のURLを参照。
http://www.microsoft.com/japan/msdn/library/default.asp?url=/japan/msdn/library/ja/cpref/html/frlrfsystemexceptionclasshierarchy.asp