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8回

ガベージコレクションの記述

ガベージコレクションとは？

.NET以前、WindowsのC/C++プログラマはリソースを自らのコードで管理していた。この管理が不徹底だと、不要になったメモリが解放されないことによってメモリリークが発生したり、解放済みリソースにアクセスしてアクセス例外が発生する。

Level

1 2 3 4 (5)

Technology Tools

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

このようにリソース管理はその記述を誤るとアプリケーションの品質に直接関わるため、慎重に実装しなければならなかったのである。しかし、メモリを適切に管理するコードの記述は非常に難度が高いので、この作業に集中していると本来注力すべきアプリケーションのロジックの実装がおろそかになることがあった。

さらに厄介なのは、これらのバグは発生を再現するのが難しいため、デバッグが相当困難になっていたことである。NuMega社（現Compuware）やRational社（現IBM）などがこれらのバグを検出するためにこれまで多くのデバッグツールを開発してきたのも、メモリ管理に依存するバグの発見がどれだけ困難であったかを示している。

ガベージコレクション（GC：Garbage Collection）は、この問題を解決する目的で開発された。開発者はメモリ管理に関する処理（リソースの確保、管理、解放）から解き放たれ、アプリケーションが本来提供する機能の実装に集中できるようになったのである。

このようにガベージコレクションは開発者にとって非常に有用な機能であるが、便利なこの機能も利用を誤るとパフォーマンスを下げるなど多くの問題を引き起こす。そこで今回は、ガベージコレクションのアーキテクチャを説明するとともに、ガベージコレクションがリソースの解放時に呼び出すFinalizeメソッド[注1]とDisposeメソッドについて、そのコードの記述方法を説明する。

ガベージコレクションのアーキテクチャ

ガベージコレクションがらみの実装で問題になるのは、やはり解放に関わる部分である。.NET Frameworkで提供されるガベージコレクションの構造上、デストラクタ（Finalizeメソッド）を普通に記述するだけでは、“ガベージコレクションが2回実行されないと解放

注1）Finalizeメソッドはアプリケーションのパフォーマンスを低下させる可能性がある。実装するケースについては「Finalizeの記述について」の項で後述。

されないクラス”になり、その分パフォーマンスが低下する。

ガベージコレクションの構造を説明するとそれだけで本連載の数回分を費やしてしまうので、ここでは「Finalizeを実装するとパフォーマンスが下がる」理由と、その解決策として「GC.SuppressFinalize(this)を呼び出す」理由を理解するために必要なアーキテクチャの知識に絞って説明する。

.NET Frameworkでは、new演算子でクラスからオブジェクトを作成するとヒープ上にメモリが確保される。このときクラスのインスタンスとして生成されるオブジェクトがFinalizeメソッドを定義していた場合、「ファイナライゼーションリスト」に登録される。ファイナライゼーションリストはガベージコレクタが制御するデータ構造で、リストのエントリはメモリが解放される前にFinalizeメソッドを呼び出す必要があるオブジェクトを示している。この構造を図で示すと図1のようになる。

図1ではA～Fのオブジェクトが生成された状態を示している。この図から、A～Fのうち“B”と“D”がFinalizeメソッドを持っていることがわかる。ここでガベージコレクションが発生すると、ガベージコレクタはファイナライゼーションリストからオブジェクトのポインタを検索し、見つかったポインタを「Fリーチャブルキュー」に追加する。Fリーチャブルキューもガベージコレクションが管理し、そのエントリはFinalizeメソッドを呼び出す準備ができたオブジェクトへのポインタを示している。図1の状態でガベージコレクションが終了すると図2のようになる。

図1：Finalizeメソッドを持つオブジェクトはファイナライゼーションリストに登録される

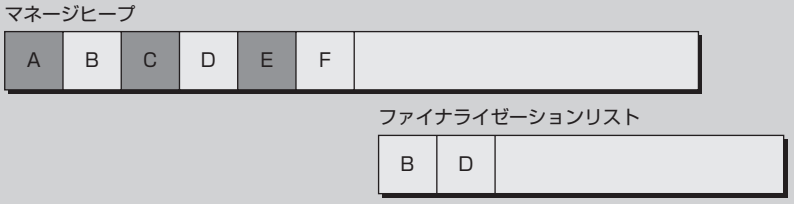
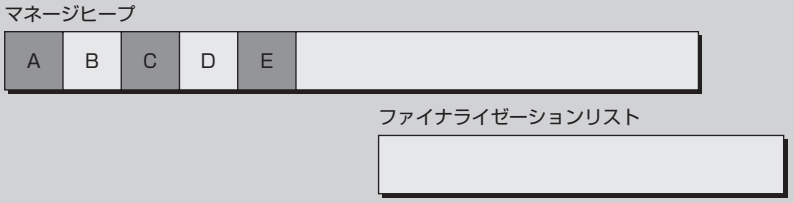


図2：図1でガベージコレクション発生／終了後の状態



“F”にはFinalizeメソッドが実装されていないので、ガベージコレクションの実行後すぐに解放される。しかし、“B”と“D”はFリーチャブルキューへ移動するだけで、まだメモリから解放されていない。この状態でもう1回ガベージコレクションが実行されると、“B”と“D”はメモリから解放される。

以上のことから、

Finalizeメソッドを実装すると、解放までにガベージコレクションが2回必要になる

ということがわかる。この意味で、Finalizeメソッドを実装するとパフォーマンスが悪くなるのである。

この対策として、MicrosoftはGC.SuppressFinalizeメソッドを用意した（GCはガベージコレクタを制御するクラス）。このメソッドを呼び出すと、thisで参照されるオブジェクトをFリーチャブルキューへ移動しなくなる[注2]ので、1回のガベージコレクションでそのインスタンスはメモリから解放されるようになる。そのため、もしFinalizeメソッドを実装するときには、必ずGC.SuppressFinalizeメソッドを呼び出す実装にしてもらいたい。

Disposeのマナー

.NETでは、ガベージコレクションによりメモリの解放漏れがなくなるが、解放のタイミングについては何の保証

注2）呼び出す場所については、「Disposeのマナー」の項で説明する。