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

最終回

パフォーマンスを分析する

発見しにくいトラブル

最近、システム開発時のテストに関する書籍や記事がIT系の雑誌に登場する機会が多くなっている。これはこの業界の成果物であるシステムに対して“動けば良い”といった甘えた考えが減ってきたことを意味している。実際、東証でのトラブルなどでシステムが業務に与える影響の大きさを目の当たりにしたユーザーの“システム品質”に対する目は厳しさを増してきている。

レベル >>> Level

1 2 3 4 5

言語 >>> Language

- C#

ツール >>> Tool

- Visual Studio .NET 2003
- DevPartner Studio Professional Edition 7.2

しかし仕様変更に追われる現場では、「テストを実行する余裕がない」→「トラブルを起こす」という悪循環を繰り返しているのが現実である。その中でもとくに問題なのがパフォーマンスのトラブルだ。仕様や画面まわりの問題は、比較的発見しやすいためリリース前に刈り取られる可能性が高いが、パフォーマンスに関する問題は、たとえソースコードレビューに時間をかけたとしてもリリースまで問題が残ってしまうケースが多い。

そこで今回は、パフォーマンス分析に絞って解説を進める。この分析方法にはログの埋め込みやパフォーマンスモニタを取得したテストなど多くの方法があるが、今回は日本コンピュウェア社の開発／デバッグ支援ツールDevPartner Studio Professional Edition（以下DevPartner）が提供する「Performance Analysis」[注1]機能を利用した解析方法について説明する。これは、一般的に「プロファイラ」と呼ばれる機能を利用した分析である。この方法は、コードを書いている開発者でも直感的に問題部分がわかるというメリットがあるので、皆さんの現場でも利用を検討してもらいたい。

注1）製品についての情報は以下のURLを参照。
http://www.compuware.co.jp/products/devpartner_fm/devpartnerstudiopro/dpspe_feature_f2.html

プロファイラの制限

分析方法の解説を始める前に、プロファイラの制約について説明しておく。プロファイラをわかりやすく表現すると、

「調査対象のコードの実行にかかる時間を、コードの実行開始から終了までの間隔を算出して測定する機能」

と言える。分析結果としてメソッドの処理時間を期待するのであれば、そのメソッドの開始時間と終了時間を測定し、差分を出すのがプロファイラの結果である。

そのため、プロファイラを利用して

も処理にかかった時間は得られるが「なぜその処理時間になったのか？」は分析できないのである。このように「遅延の原因が分析できない」という理由から、プロファイラによる分析を重要視しない人が現場で多く見られる。しかし、パフォーマンスに関するトラブルの分析方法として、どの部位の処理が遅いのかがすぐに特定できるプロファイラはもっと利用が検討されてもよい機能なのである。

この制約を理解し、まずはじめにプロファイラでパフォーマンスに問題が出る処理を特定してから、システムリソースの処理状況を調査するなどのチューニング作業を行なうのが、パフォーマンス問題の対応として最も効率の良い方法であると言える。

実行状態を診断する

では早速、プロファイラによる分析を説明する。

DevPartnerを持っていない人は日本コンピュウェア社の以下のサイトから評価版がダウンロードできるので入手して試してもらいたい。

http://www.compuware.co.jp/products/devpartner_fm/dl_devpartner.html#dpsp72

今回用意したのは、リモートマシンのデータベースにアクセスして得られたデータをソートする簡単なコードである（記事末リスト1）。コードがビルドできたら、分析を開始する。

DevPartnerをインストールすると、図1のようにVisual Studio .NET（以下VS.NET）のツールバーとして提供機能が表示される。この中の時計マークのボタンを押して「Performance Analysis」機能を開始する（または、VS.NETのメニューから［ツール］-［DevPartner］と移動して［Performance Analysis］を選択）。

図1：VS.NETのツールバーからDevPartnerの機能を起動できる

こうしてプロファイラ機能をスタートした後、［F5］キーを押して測定対象のコードをVS.NETから起動する。

そして測定対象の処理終了後、デバッグ実行を止めて測定を終了する。

測定終了後、VS.NETのソリューションエクスプローラに図2のように処理結果が表示される。「<プロジェクト名>.dpprf」が測定結果を記録したファイルである。このファイルを開くとVS.NETの画面上に、

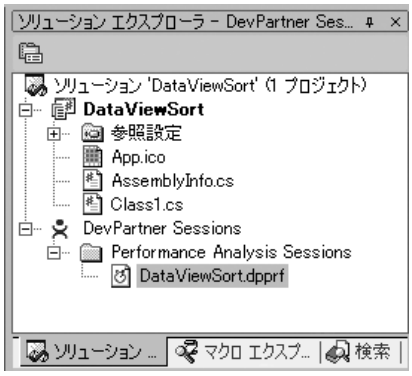
- Method List
- Source［ファイル名］
- Session Summary

という3種類の情報がそれぞれタブで表示される。解析に一番有用なのは「Source」タブの内容なので、これをクリックして解析結果を表示する。

図3のように「Source」タブには4種類の情報（表1）が表示される。この解析結果を利用して簡単に解析方法を説明する。

「Source」タブを利用したボトルネック部位の特定は以下の手順で行なわれる。

図2：プロファイラの測定結果をファイルとして生成



手順1　「Time」を見て各処理の時間を調査

手順2　処理の長いメソッドの実装部分で「Time」を調査

手順3　システム関数または他マシンへのアクセスなど、それ以上分析できない部位まで追跡調査

この順番で解析を行ない遅延を起こしている部位を特定する。

ここで図3に示したサンプルコードの結果を利用して調査してみよう。はじめに注目すべき点は「Time」である。処理時間の長かったコードは赤色で表示される（それ以外は黒色で表示）。なお、図3では色の違いがわかりにくいため、赤色で表示された部分を罫で囲んでいる。