遅い遅いと嘆くより、進んでコードを見直そう

Windowsアプリケーションの高速化／最適化

あらゆるテクニックを駆使してパフォーマンスの極限に挑む！

マイクロソフト株式会社
ビジネスパートナー
営業本部SE部
平井 昌人
HIRAI, Masato

Technology Tools

- ☑ Visual Basic .NET
- ❑ Visual C# .NET
- ❑ SQL Server 2000
- ❑ Oracle 9i
- ❑ Access 2002
- ❑ ASP.NET
- ❑ Internet Information Services
- ❑ Other:

Level

★ ★ ★ ★ ★

Samples

・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、付録CD-ROMの¥DOTNET¥F01ディレクトリに収録しています。

¥01NGENDEMO
プリコンパイルによる速度差実験サンプル
¥02RESOURCEDEMO
リソースの格納位置に関するサンプル
¥03ADDRANGEDEMO
AddとAddRangeの速度差実験サンプル
¥04BEGINUPDATEDEMO
BeginUpDateの使用サンプル
¥05RESIZEDEMO
リサイズ時の速度差を視認するサンプル
¥06IMAGESANDLINES
自動スケーリング実験サンプル
¥07QUALITYTEST
クオリティが速度に与える差の実験サンプル
¥08ROTATETEXT
ダブルバッファリング実験サンプル
¥09EASYINVOKE
同期／非同期実験サンプル
¥10STRINGPERFORM
ガベージコレクション実験サンプル
¥11WEAKREF
フォームの再表示を高速化するサンプル
¥12ONERRORDEMO
On ErrorとTry Catchを比較するサンプル
¥UTY
本稿で使用したユーティリティアプリケーション
README.TXT
始めにお読みください

ご挨拶

みなさん、こんにちは。マイクロソフト株式会社 SE部の平井昌人です。今回はdotNETマガジン編集部の並々ならぬ熱い願いとしつこいまでの説得攻撃に屈して久々に寄稿させていただくことになりました。テーマは「Windowsアプリケーションの最適化」。

これは2004年の夏、横浜で開催したMicrosoft Tech・Ed 2004というテクニカルカンファレンスで筆者が講演させていただいたセッション『T5-407：Windowsフォームにおけるパフォーマンス向上の実現』の内容をそのまま記事にしています。別にラクをしているわけではないです。セッションのスライドやデモを作るよりも文書を書くほうがとっても面倒なのです。どこかで聞いたこと、見たことがあるなぁと思う方もいるかも知れませんがご勘弁ください。

本稿は、主に.NETでクライアントアプリケーションの開発に従事されている中／上級のプログラマの方を対象にしています。このため、内容もちょっぴり濃い目に技術レベルも少し高めに設定してあります。

はじめに

パフォーマンスチューニングと言えばデータベースやサーバーコンポーネントなどがメインで我々はこの作業に莫大な時間とコストをかけています。たしかにデータベースやサーバーコンポーネントは、常に高いパフォーマンスとスケーラビリティが要求されるため、最適化のやり甲斐もあるでしょう。しかし、サーバーサイドのパフォーマンスが向上してもそれを利用するWindowsクライアントアプリケーションのパフォーマンスや応答性が悪ければ、システム全体の価値が低減してしまうのも事実です。これからのWindowsクライアントアプリケーションに求められるものはリッチでスマート。クライアント開発者は、華麗でグラフィカルなユーザーインターフェイスを提供しつつ、操作性やパフォーマンスを最大限に引き出すことを考慮して設計／実

装していかなければなりません。今回は、Windowsフォームを使ったクライアントアプリケーションでのパフォーマンスの改善に焦点を絞り、数々のテクニックやノウハウを具体的なコード例とサンプルで実証してゆきます。

サンプルに関する注意事項

サンプルはすべてMicrosoft Visual Studio .NET 2003 （VB.NET）で記述してあります。動作手順については各フォルダにある「デモスクリプト.txt」をご覧ください。収録したサンプルは筆者が個人的に作成したものでマイクロソフトから正式に提供されるものではありません。また、本稿に記載したパフォーマンスデータは表1のマシン環境で測定したもので環境により多少異なる場合もあります。

なお、サンプルプログラムで使用している画像ファイルの一部（写真画像）は、株式会社インプレスから刊行されている『Internetホームページ用素材集Photoコレクション』に含まれるものを利用させていただきました（Original Photo by XS Software Inc.）。このサンプルに含まれる画像ファイルは、ライセンスの関係で二次配布ができません。したがってサンプルに含まれるすべての画像ファイルの複製、再配布を禁止させていただきます。

表1：サンプルプログラムの検証環境

CPU	Intel Pentium4 2.8GHz / FSB800
Memory	1024MB
OS	Windows XP SP2
.NET Framework	1.1

COLUMN 筆者自身による自己紹介

1993年4月に Quick Basic、MS-BASICのサポートエンジニアとしてマイクロソフトに入社。もうMSに在籍して11年となる。現在はシステムエンジニア（いわゆるSE）としてISVや開発者向けに.NETの技術啓蒙や開発案件支援を行なっている。また、数々のセミナーやカンファレンスのスピーカーに借り出されている。出張も多く、多忙な日々を送っているわりには、体重と体脂肪率の増加に歯止めがかからない。現在、0x28歳。年老いてもまだまだプログラミング力は落ちていない。それどころか年々レベルアップをしている。自称“天才プログラマ・ジニアス平井”。あれ、ドン引きですか!?　ところで最近は何とあのブロックのLEGOにハマっている。LEGOブロックで自宅PCのケース作りに挑戦中。LEGOの青いバケツがたくさん必要でマザーやHDDなどのパーツを固定するのが結構難しい。

プリコンパイルによりパフォーマンスを向上させる

.NETは実行時コンパイル

はじめにアセンブリのプリコンパイルの機能を解説する。

プリコンパイルはプログラムの修正や再ビルドの必要がないためパフォーマンス向上の一番楽な手法といえる。VB.NETやC#でプログラムを作成してビルドすると実際のコードは、MSIL（Microsoft Intermediate Language）と呼ばれるCPUやOSに依存しない中間言語にコンパイルされる。図1のようにVB.NETとC#で加算を行なうコードを記述してアプリケーションをビルドしてみよう。次に生成された実行ファイルを.NET Framework SDKツールの“ILDASM.exe（MSIL逆アセンブラ）”で逆アセンブルするとまったく同じMSILが生成されていることがわかる。つまり、MSILという中間言語の状態でアセンブリ（EXEやDLL）の中に収められているわけだ。そして、図2のようにこのアプリケーションが実行される（または別のアセンブリから呼び出される）とCLR（共通言語ランタイム）のクラスローダーが必要なクラスをメモリに読み込み、JIT（ジャストインタイムコンパイラ）がそのクラスのMSILをCPUやOSに最適なネイティブコード（現在はx86 32bitコード）にコンパイルする。

この仕組みを「実行時コンパイル」という。JITはメソッドやプロパティが呼ばれた段階でそのつどコンパイルを行なうため、効率がよくパフォーマンスも最適化されている。もちろん、利