.NET Frameworkで作るWindowsサーバー

作ればわかるアプリケーションの動作とメカニズム

第12回 .NET Framework 2.0で作る非同期通信ソケットサーバー

秋月巌ソリューション事務所
秋月 巌 AKIZUKI, Iwao
http://www.akizuki.co.jp/

Windows Server

非同期通信の単一スレッドサーバー

本連載の第5回（2005年9月号）で非同期通信を扱ってから、前回のPOP3サーバーまで非同期通信のサーバーを作ってきた。第8回（2005年12月号）で非同期通信のマルチスレッドサーバーをテンプレート化したものを「AMT（Asynchronous Multi Threading）サーバー」と称し、それを利用してきたのである。

サーバープログラミング支援のためのテンプレートであるAMTサーバーは、コンポーネントによる機能提供と異なり、プログラミングスタイルがほとんど変化しないので、本記事のように技術紹介が目的のものには最適だった。Socketクラスを使った通常のプログラミングスタイルと同じままで、プロトコルの実装のような新しく追加実装された部分をイベントプロシージャとして切り離すことができたからである。ちなみに、このイベントプロシージャはAMTサーバーの「App」で始まる名前を持つプロシージャ群である。開発者はこの「App」で始まるプロシージャにプログラムコードを記述することでプログラムを実装できる。実際にはSocketクラスの非同期通信でコールバックされるプロシージャは別のプロシージャなのだが、最終的にはこれら「App」で始まるプロシージャが呼び出されるのである。

もちろん、新しくクラスを開発して同じ機能を持たせることも簡単だが、技術記事を読むために、他者が作ったクラスの使用方法やプログラミングスタイルの学習をしたくない読者も多いだろうから、AMTサーバーというテンプレートは有用だったと思う。

しかし、AMTサーバーテンプレートは、.NET Framework 2.0ではマルチスレッド時にエラーが発生したり、また、Socketクラスがスレッドセーフでないなど不安要因もある。それに実装が複雑過ぎて、作った本人ですら、ときどき混乱してしまうようなものを技術サンプルに用いるのは、問題があるかもしれない。それにマルチスレッドが性能的に有利な場面もかなり限定されるので、今回は単一スレッドで動作する「AST（Asynchronous Single Threading）サーバーテンプレート」を提供して解説する。このASTサーバーテンプレートはAMTサーバーテンプレートと

レベル >>> Level
1 2 3 4 5

言語 >>> Language
- Visual Basic

ツール >>> Tool
- Visual Studio 2005

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。

完全互換のプログラミングインターフェイスを備えているので、AMTサーバーテンプレートで作ったプログラムを移植するには、単純にプログラムコードをコピーすればいい。

また、今回は非同期通信クライアントプログラムを作るためのテンプレートであるAsyncクライアントも提供する。今まで、本連載ではWebブラウザやメールクライアントのような既存のクライアントアプリケーションからアクセスするためのサーバープログラムを作ってきたが、今後は専用クライアントを持つサーバーを作る可能性もあるので、役に立つだろう。

今回の記事で得することと約束ごと

さて、本記事で提供するサンプルとテンプレート（AMTサーバーとAsyncサーバー）を理解すれば、読者の方は.NET FrameworkのSocketクラスを理解するより少ない学習量で非同期通信プログラムを作ることができる。とはいえ、よくあるコンポーネントにありがちな特殊な学習ではなく、Socketクラスのプログラミングスタイルそのままの形で学習できるので、学習したことをそのまま、Socketクラスのプログラミングに役立てることができる。

また、サーバー側では接続中のユーザーを配列として管理しているので、タイムアウトの管理や全員にデータを送信するなどの機能をプログラミングせずに実装することができる。Socketクラスを最初から学習してプログラムを作るよりも、お勧めである。すでにSocketクラスによる非同期通信プログラムの学習経験のある方ならば、テンプレートのコードをとくに苦労せずに読むことができると思うので、やはり、お勧めである。ソースコードは完全に公開されているので、気に入らないところがあれば直せばいい。私はいままで、同じアーキテクチャのAMTサーバーテンプレートを使って、Webサーバーとメールサーバーを作ってきて、通信コンポーネントとして妥当な設計なのではないかと思っている。もちろん、実装は技術サンプルの領域を出るものではない（たとえばエラーハンドリング）が、それは使う人が、必要に応じて実装すればいいだろう。一般的なコンポーネントが高機能で厚いレイヤーを持つのに対して、今回提供するテンプレートは、低機能で薄いレイヤーであることを特徴とする。

ここまでで、興味を持った方は理解を早めるために、次の規則を最初に理解しておいてほしい。

規則1：「_」（アンダースコア）で始まる変数は、開発者がプログラムで設定可能なプロパティである。また、「_」で始まるプロシージャは、開発者が自分で作ったプログラムから呼び出して使えるメソッドである。

規則2：「App」で始まるプロシージャは、開発者がプログラムを記述できるイベントプロシージャである。「CallBack」で始まる同名のコールバックプロシージャから呼び出されるものが多い。つまり、通常、コールバックプロシージャに記述されるコードをここに記述すればいい。

これらのプロシージャ、変数の全一覧が表1である。

フォームと同一のスレッドで動作させるか否か

さて、単一スレッドといっても、フォームが動作しているメインスレッドで通信プログラムを動作させるか、あるいは通信部分はフォームとは別の単一スレッドで動作させるかで実装に違いがある。ちなみにマルチスレッドのAMTサーバーは通信部分自体を複数のスレッドに分離させる構造になっていた。つまり、あるクライアントと別のクライアントはサーバーの別のスレッドで通信する可能性があるというものだった。

単一スレッドのASTサーバーテンプレートの通信をメインスレッドで行なった場合、それはあらゆる意味でシングルスレッドプログラムだし、メインスレッドとは別のスレッドで通信を行なった場合、それはプログラムとしてはマルチスレッドプログラムということになる。まぁ、どちらでも大した差はないのだが、どちらにするべきかの明確な指針が得られなかったので、両方を用意することにした。

そこで、今回のサンプルは次のように4つの構成となる。