.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6回 “マルチスレッド”非同期通信ソケットサーバーを作る

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

level

1 2 3 **4** 5

Technology Tools

- ☑ Visual Basic
- ☐ Visual C#
- ☐ Visual C++
- ☐ SQL Server
- ☐ Oracle
- ☐ Access
- ☐ ASP.NET
- ☐ Other:

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、http://www.shoeisha.com/mag/windev/からダウンロード可能です。

非同期ソケットサーバーをマルチスレッド化

前回は単純な非同期ソケットサーバーの作り方を解説した。今回は、それをマルチスレッド化する。非同期ソケットならば非同期処理ができるのだから、マルチスレッド化する必要はないと考えるのが妥当かもしれない。しかし、マルチスレッド化することでマルチプロセッサの恩恵を受けやすいというメリットがある。とはいえ、非同期ソケットサーバーをマルチスレッド化する作業は、それほど単純ではない。非同期ソケットサーバーだけでも、かなり混乱しやすいのに、それにマルチスレッドのためのプログラミングが加わるのである。そこで本稿では、前回説明した非同期ソケットサーバー処理を理解しているという前提で解説する。

今回は以下のようなサンプルプログラムを用意した。

サンプル1
ManualResetEventクラスを理解するための簡単なプログラム

サンプル2
接続とデータの送受信を別スレッドで行なう非同期通信プログラム（データの送受信は単一スレッド）

サンプル3
接続とデータの送受信を別スレッドで行なう非同期通信プログラム（データの送受信もマルチスレッド）

ManualResetEventクラスを理解する

非同期ソケットの知識の他に、今回作成したサンプルプログラムを理解するには、前回も紹介したManualResetEventクラスの利用法を理解する必要がある。サンプル1のプログラム（リスト1・図1）は、ManualResetEventクラスを理解するための簡単なサンプルプログラムである。真中のボタンをクリックすると、5秒後にメッセージボック

リスト1：ManualResetEventオブジェクトを利用する簡単なプログラム（サンプル1）の全ソースコード

```
Public Class Form1
  Inherits System.Windows.Forms.Form
  Private WorkThread As Thread
  Private MREvent As New ManualResetEvent(False)

 Windowsフォームデザイナで生成されたコード

  Private Sub btnStart_Click( _
    ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnStart.Click
    WorkThread = New Thread(AddressOf MainProc)
    WorkThread.Start()
    MREvent.WaitOne()
    MREvent.Reset()

    MsgBox("「ManualResetEvent」クラスのSetメソッド" & _
           "が実行されました。")
  End Sub

  Private Sub MainProc()
    Thread.Sleep(5000)
    MREvent.Set()
  End Sub
End Class
```

スが表示される。

ボタンがクリックされると別のスレッドを起動し、その子スレッドは5秒間、処理を待機する。メインスレッドは待機せずに処理を継続するが、ManualResetEventクラスのWaitOneメソッドで停止する。5秒後、待機終了した子スレッドでManualResetEventクラスのSetメソッドが実行される。これがWaitOneメソッドで停止しているメインスレッドに通達される。これで以降の処理が実行され、ダイアログが表示されるのである。

このようにManualResetEventオブジェクトを使って、他のスレッドの実行を制御することができる。この方法をサンプル2以降では多用する。元々、非同期通信のプログラムは実行の順序がわかりにくいが、これによりさらにわかりにくくなってしまう。他にもっとよい方法があるのかもしれないが、非同期マルチスレッドの資料は見つからなかったので、私はこのManualResetEventオブジェクトを使用して、他のスレッドを制御する方法で実装した。

ちなみにManualResetEventクラス同様、WaitHandleクラスを継承したクラスにAutoResetEventがある。AutoResetEventクラスを使ってもこのサンプルの場合、まったく同じように動作する。AutoResetEventクラスを使った場合、このサンプルでWaitOneメソッドの直後に実行されているResetメソッドを実行する必要がない。つまり、AutoResetEventクラスとManualResetEventクラスの違いはWaitOneメソッドの後に自動的にResetメソッドが実行されるか否かの違いだと考えることができる。

ちなみに、ManualResetEventクラスを使ったこのサンプルでResetメソッドの行をリマークすると、1度目は

図1：サンプル1実行画面

正しく動作するが、2度目以降は、ボタンクリック後、すぐにメッセージボックスが表示されてしまう。

これらのクラスのインスタンスが作成された直後、オブジェクトの状態は初期状態である。初期状態でWaitOneメソッドを実行するとスレッドの実行が停止する。他のスレッドで同じオブジェクトのSetメソッドを実行すると、スレッドの実行が再開する。このときAutoResetEventオブジェクトならば自動的にオブジェクトの状態をリセットし初期状態に戻すし、ManualResetEventオブジェクトならばそのままである。オブジェクトをリセットしないでWaitOneメソッドを実行した場合、スレッドは停止しない。

ひとつのManualResetEventオブジェクトに対して2つ以上のスレッドでWaitOneメソッドを実行することも可能である。この場合、別のスレッドがSetメソッドを実行すると、ひとつのスレッドだけが実行を再開する。別のスレッドのWaitOneメソッドを解放するためには、さらにSetメソッドを実行する必要がある。

また、同じ親クラスを持つクラスにMutexクラスがある。このクラスは初期状態でWaitOneメソッドを実行してもスレッド（第1スレッド）は停止しない。ただし、そ