特集2

ふたつの技術の使い分け指針

リモーティングとWebサービス

日本ユニシス株式会社
.NETビジネスディベロプメント
尾島 良司 *OJIMA, Ryoji*

みなさん、.NET Remotingでオブジェクト間通信していますか？　昔、私が日本ユニシスに入社したころは、コンピュータ間通信には、TCP/IPのソケットライブラリを使って独自プロトコルの電文を作成したり解釈したりという非常に大変な作業が必要でした。でも、そんな苦労も今は昔。分散オブジェクト技術の登場により、コンピュータ間通信ははるかに簡単に実装できるようになりました。それどころか、現在は分散オブジェクトの実装系が複数あって、いったいどれを選べば良いのかがわからないという、なんとも幸せな状況だったりします。さて、.NET Framework使いの我々にとって、その選択肢は.NET RemotingとXML Webサービスの2つ。幸せな悩みだとはいっても悩みは悩み。本稿では、このどちらを使えば良いのかについて考えてゆきましょう。

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¥F02ディレクトリに収録しています。

¥SIMPLEREMOTING
簡単な.NET Remotingのサンプル

¥SOCKETCOMMUNICATION
ソケットを使用したサンプル

¥LOGCHANNELSINK
カスタムSinkを使用してログを取得するサンプル

まずは.NET Remotingしてみよう

何かと話題になることが多いXML Webサービスと比べて、.NET Remotingは情報が少ないように感じられます。.NET Remotingを使ったことがないという方も多いのではないでしょうか？

というわけで、まずはウォーミングアップ。簡単な.NET Remotingのプログラムを作ってみることにしましょう。

なお、本稿で使用したサンプルはすべて、付録CD-ROMに収録されています。ぜひ、実際に動かしてみてください。

サーバーコンポーネントを作成

¥SimpleRemoting¥ServerComponent

.NET Remotingは、異なるコンピュータやプロセス、AppDomainに存在するオブジェクトを呼び出すことを可能にします。

ただし、.NET Remotingを使えば外部のどんなオブジェクトでも呼び出せるわけではありません。.NET Remotingで呼び出されるクラスは、少しだけ特殊な作り方をしなければならないのです。

とは言っても作業は簡単。サーバーとなるクラスはMarshalByRefObjectを継承し、引数や戻り値に使用するクラスにはSerializable属性を追加するだけ。簡単でしょ？　さっそくやってみましょう。

リスト1がサーバーとなるクラス、リスト2が引数や戻り値に使用するクラスのソースコードです。MarshalByRefObjectを継承しているところと、Serializable属性が付加されている以外は、ごく普通のクラスであることがおわかりいただけたと思います。

あとは、この2つのクラスを、クラスライブラリとしてコンパイルしてください。これらのクラスは、.NET Remotingのホストアプリケーションやクライアントアプリケーションから参照する必要があるためです。

リスト1：サーバーとなるクラス（GreetingServer.cs）

```
using System;

namespace ServerComponent
{
  // .NET Remotingのサーバーは、MarshalByRefObjectを継承する
  public class GreetingServer: MarshalByRefObject
  {
    public Greeting SayHello(string name)
    {
      Console.WriteLine("GreetingServerが呼び出されました。");
      return new Greeting("こんにちは、" + name + "さん。");
    }
  }
}
```

ホストアプリケーションを作成

¥SimpleRemoting¥HostApplication

さて、先ほど作ったサーバーコンポーネントですが、このままでは.NET Remotingで呼び出すことはできません。

.NET Remotingの呼び出しを監視し、受け取った要求を先ほどのサーバーコンポーネントに転送するホストアプリケーションが必要なのです。次は、このホストアプリケーションを作成してみましょう。

とは言っても作業はやっぱり簡単。.NET Frameworkは、呼び出しを監視し、受信した呼び出し要求をサーバーコンポーネントへ転送する機能を持っています。よって、必要な作業は環境の設定だけ。しかも、環境を設定するための構成ファイルのスキーマも定義されていますから、構成ファイルを作って環境設定をするAPIを呼び出すだけとなります。

では、具体的なコードを。ホストアプリケーションのソースコードをリスト3、構成ファイルをリスト4に載せます。以下、その中身を見てゆきましょう。

ソースコード

リスト3のMainルーチンでは、RemotingConfigurationクラスのstaticなメソッドであるConfigureを呼び出しています。Configureの引数は構成ファイルの名前。.NET Remotingの構成ファイルの名前は本当は何でも良いのですけど、Visual Studio .NETには「App.config」というファイルを「アプリケーション名.exe.Config」という名前にして出力ディレクトリにコピーする機能がありますから、

アプリケーション名.exe.config

リスト2：引数や戻り値に使用するクラス（Greeting.cs）

```
using System;

namespace ServerComponent
{
  // .NET Remotingで引数や戻り値となるクラスには、
  // Serializable属性を付加する
  [Serializable]
  public class Greeting
  {
    private string message;

    public Greeting(string message)
    {
      this.message = message;
    }

    public string Message
    {
      get { return this.message; }
    }
  }
}
```

リスト3：ホストアプリケーション（HostApplication.cs）

```
using System;
using System.Diagnostics;
using System.Runtime.Remoting;
using ServerComponent;

namespace HostApplication
{
  public class HostApplication
  {
    public static void Main()
    {
      // .NET Remotingの初期化
      RemotingConfiguration.Configure(
        Process.GetCurrentProcess().ProcessName +
        ".exe.config");

      // ひたすら待つ
      Console.ReadLine();
    }
  }
}
```

にすることをお勧めします。

あとは、ホストアプリケーションが終了してしまわないようにConsole.ReadLine()でひたすら待つようにします。これでホストアプリケーションは完成です。

構成ファイル

リスト4をご覧になればわかるように、残念なことに構成