コンポーネント化のメリットと手法を追求する

Visual Studio .NETで究めるコンポーネント開発

第4回 オブジェクトのシリアライズ

株式会社コムラッド
辻 慶 *TSUJI, Kei*
http://www.comrade.co.jp/

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¥NETARCH03ディレクトリに収録しています。

¥SERIALIZETEST
シリアライズ／デシリアライズの確認
¥PASSWORDHOLDERSERIALIZETEST
SerializableAttribute問題点1
¥INCORRECTSTATICFIELDSERIALIZETEST
SerializableAttribute問題点2
¥STATICFIELDSERIALIZETEST
Staticフィールドのシリアライズ
¥MULTIPLESTATICFIELDSTEST
複数のStaticフィールドのシリアライズ
¥REFLECTIONTEST
リフレクションを使用した、文字列による型の取得
¥REFLECTIONANDINTERFACE
リフレクションとインターフェイスの使用例

はじめに

前回はカスタムクラスの設定をデザイン時に変更する方法について説明しましたが、記事の最後で触れたように、このままでは設定の変更が反映されません。設定値を永続化させる仕組みを実装していなかったからです。

そこで今回は永続化の具体的な説明に入る前段階として、独自クラスを永続化する際に必要となる“シリアライズ”について説明することにします。

シリアライズとは？

「シリアライズ」とは、メモリ上にある情報の一部、あるいはすべてを、ファイルとして保存したりネットワークで送受信したりできるように変換することをいいます。反対にシリアライズされたデータから、ソフトウェアで扱うことのできる元のデータ形式に直し、メモリに復元することを「デシリアライズ」といいます。これを広義のシリアライズと呼ぶことにしましょう。その場合、Visual Studio .NET（以下VS.NET）のコード生成もシリアライズに該当します。

ただし、ソフトウェアの世界で“シリアライズ”と言えば、一般的には「バイナリシリアライズ」を指す場合が多いでしょう。このほか、XMLシリアライズなどもあります。

.NET Frameworkには、バイナリ／XMLの両方でシリアライズ／デシリアライズを簡単に扱える仕組みが備わっていますが、本稿ではXMLでのシリアライズに関しては扱いません。以降では、バイナリシリアライズに関して説明してゆきます。

バイナリシリアライズ

バイナリシリアライズを実装するのは比較的難しい部類に入ります。しかし、前述のとおり、.NET Frameworkに備わっている仕組みを利用することにより、簡単に実装することができます。

もっとも簡単な方法はシリアライズ

リスト1：シリアライズ可能なクラス

```
[Serializable]
public class SerializableClass {
   int data;
   public SerializableClass(int data) {
      Data = data;
   }
   public int Data {
      get { return data; }
      set { data = value; }
   }
}
```

リスト2：シリアライズ／デシリアライズの確認

```
const string FILENAME = "test.bin";
SerializableClass sc = new SerializableClass(2), deserialized_sc;
BinaryFormatter bf = new BinaryFormatter();
using (FileStream fs = File.OpenWrite(FILENAME)) {
   bf.Serialize(fs, sc);
}
using (FileStream fs = File.OpenRead(FILENAME)) {
   deserialized_sc = (SerializableClass)bf.Deserialize(fs);
}
Console.WriteLine("Original : {0}, Deserialized : {1}",
                  sc.Data, deserialized_sc.Data);
```

したい型に対して、「Serializable」Attributeを指定する方法です。簡単なサンプルをリスト1に示します。これを見るとわかるように、単純にSerializableAttributeを指定するだけで実装できます。

次に、このクラスがきちんとシリアライズされ、シリアライズされたデータが元通りにデシリアライズされるかどうかを確認する方法を見てゆきましょう。

今回はバイナリシリアライズの確認なので、「BinaryFormatter」を使用します。ここでは詳しく説明しませんが、SOAPシリアライズの場合はSoapFormatter、XMLシリアライズの場合はXmlSerializerを使用します。

BinaryFormatterのSerializeメソッドでは、シリアライズされたオブジェクトをStreamに流すことになります。Streamにはもちろん MemoryStreamや他のStreamも使用可能ですが、FileStreamを使用するとバイナリエディタなどで中身を確認することができるので、今回はFileStreamを使用しています。

リスト2に簡単な確認のコードを示します。このコードを利用すると、上記のように、BinaryFormatterのSerializeメソッドで、SerializableClassがFileStream中にシリアライズされ、次にFileStreamからきちんとデシリアライズされていることを確認できます（図1）。

SerializableAttributeの問題点

このようにSerializableAttributeは、簡単にシリアライズ／デシリアライズを行なうことができる便利なものです。ほとんどの場合は単にSerializableAttributeを指定するだけで十分です。ただし、SerializableAttributeを使用する場合

コマンドラインによるコードの確認方法

まず、サンプルソースの入ったディレクトリを、わかりやすい位置にディレクトリごとコピーします。

次に、OSの［スタート］メニューから［プログラム］-［Microsoft Visual Studio .NET 2003］-［Visual Studio.NETツール］-［Visual Studio .NET 2003コマンドラインプロンプト］を選択し、VS.NETのコマンドラインプロンプトを実行します。そして、目的のディレクトリ（ここでは“C:¥SerializeTest”）で、

```
C:¥SerializeTest> csc SerializeTest.cs
```

と入力すると、VC#コマンドラインコンパイラによってコンパイルされ、“SerializeTest.exe”というアセンブリが生成されます。生成後、

```
C:¥SerializeTest> SerializeTest.exe
```

と入力すれば、EXEファイルを実行して確認することができます。コマンドラインコンパイラのオプションについての詳細は、

```
C:¥> csc /?
```

と入力すると表示されます。