特集2

Domain ModelからDataSetへ

こんにちは、Table Module。ようこそ、DataSetの世界へ

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*

引越ししました。現在、どこに何が入っているかがまったく不明なダンボールに囲まれて、小動物のような感じでオドオド生きています。ヤバイ。どこかのダンボールに食料品が入っていたような気がします……。
で、会社でこの話をしたら、最近の引越し業者はスゴイと教わりました。なんでも、どこの棚のどの辺りに入っていたものかまでラベル付けしてくれて、完全に元のとおりに戻してくれるのだそうです。その人によれば、ピカチューのオモチャを入れた箱には大きくピカチューの絵が描かれ、しかもスイッチを切り忘れたらしくて引っ越し中はどこからともなく「ぴっかちゅうぅ〜」という声が聞こえてくる素敵な引越しだったそうな。
そう、コンピュータのシステムでもこれは同じ。RDBMSから取得したデータは、わかりやすい形で正しくビジネスロジックに渡さなければなりません。ビジネスロジックが加工した結果も、やはりわかりやすい形で正しくGUIに渡さなければなりません。本稿では、このデータを運ぶ箱であり、.NET Frameworkを特徴付けている“DataSet”について述べさせていただきます。

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。

・LIST1〜6.TXT
リスト1〜6のコード

.NET Frameworkへの道程

申し訳ありませんが、ちょっと遠回りして通信の歴史の話をさせてください。歴史を理解することは物事の本質を理解する遠回りに見える近道なので、どうかご容赦を。

では、「電文時代」→「ミドルウェア全盛期」→「言語の高度化」→「.NET Framework」の順で見てゆきましょう。

電文時代

大昔、ホストコンピュータでは2,400bpsの専用回線[注1]で、FEP（Front End Processor）といえば仮名漢字変換ではなくて通信制御のハードウェアだったころ。

その専用回線の中でデータを運んでいたのは“電文”と呼ばれるだらだらした文字列で、その電文を生成したり解釈したりするのはプログラムの役割でした。

もちろん、そのプログラムは我々プログラマが作らなければならないわけで、それがもう地獄のような作業でした[注2]。だって、だらだらした文字列という以外の構造を持たないわけですから、その中にどのような構造を作るのかは設計者の気分次第なのです。その“気分”を分厚いドキュメントの中から抽出してプログラミングする。しかも繋げて実際に動かすまで正しく実装できたかわからない。

こんなの、誰だって嫌ですよね？

ミドルウェア全盛期

というわけで、だらだらした文字列という電文はやめて、プログラムからのアクセスが容易な“型”を準備し、

注1）ちなみに、その頃の私は300bps（56kbpsモデムの1/180の速度）の音響カプラなマイコン少年でした。
注2）ホスト連携でよく出てくる、固定長フォーマットの作成と解釈を思い浮かべてください。

その型のインスタンスをやり取りする方式が編み出されました[注3]。CORBAやCOMのIDLのstructがそれ。この方式なら、我々プログラマもまぁ納得です。

この通信で使う型を通常のロジックでも使うようになるのは、当然の結果でしょう。GUIは通信で取得した型を表示／作成し、ビジネスロジックはこの型に対するロジックを実行します。これでさらに実装が楽になりました。

でも、IDLのstructにはフィールドしか設定できなかったので、通常のロジックで使うにはイマイチ使い勝手に不満がありました。オブジェクト指向のカプセル化が使えないわけで、フィールドの解釈や設定は自分でやらないとならなかったのです。これでは電文とあまり変わりません。

ではどうしましょうか？　IDLのstructが使いづらいなら、メソッドやプロパティを定義できるclassを使えばよいわけです[注4]。というわけで、CORBAは引数や戻り値にclassのインスタンスを使えるようにするObject-by-Valueという機能をサポートしました。

言語の高度化

そして、ついにJavaなどのネイティブに通信とシリアル化をサポートする言語が登場しました。普通に作ったclassのインスタンスを、そのままネットワークに流すことができるようになったわけです。

こうなると、わざわざCORBAなどというミドルウェアを後付けする必要はなくなってしまいます。ミドルウェアの時代はここに終わりを告げ、CORBAやObject-by-Valueという言葉は表舞台から姿を消したのでした[注5]。

.NET Framework

さて、このような良い時代になっても、人間の欲望には限りがありませんでした。

シリアル化されてネットワークを流れているデータとか、ネットワークに流す代わりに記憶装置に格納したデータとかを見たいという話が出てきたのです。また、さまざまなOSや言語のプログラムとも安価に通信したいという要求も出てきました。

幸いなことに、当時すでにこれらの要求を完全に満たすXML（eXtensible Markup Language）という道具がありました。あとはこのXMLを用いてSOAPというRPC（Remote Procedure Call）の規格を作成するだけ。これで通信の問題は解決です。

残る問題は言語。XMLによるシリアル化に対応した言語が必要とされていたのです。ミドルウェアによるアプローチよりも言語によるアプローチのほうが優れていることは、CORBA vs. Javaで証明されていましたから。

ここで、我々の期待に応える形でMicrosoftが“.NET Framework”を発表しました。そう、このXMLによるシリアル化にネイティブ対応した言語こそ、.NET Frameworkなのです。ふぅ、やっと繋がった。

XMLに対応していると、開発や運用が簡単になります。XMLはただのテキストなので、既存のあらゆるソフトウェアと組み合わせることができます。ログとしてバイナリのダンプを見せられても意味不明ですけど、XMLならすぐに中身がわかります。XMLはあらゆる環境でサポートされていますから、他の環境との相互作用も完璧です。

いやぁ、.NET Frameworkって素晴らしいですな。

ビジネスロジック実装の3つのパターン

とまぁ、かように便利な.NET Frameworkですが、どんなに優れた道具があっても、使い方を間違えちゃったら意味がないわけです。

というわけで、.NET Frameworkの使い方の選択肢として、Martin Fowlerの『Patterns of Enterprise Application Architecture』から、「Transaction Script」と「Domain Model」「Table Module」をご紹介しましょう（図1）。

Transaction Script

構造化手法で処理を分割して実装するやり方です。共通部分は構造化のアプローチで導きます。

Transaction Scriptにはプログラミングモデルがシンプルだというメリットがあります。ビジネスロジックがシンプルな場合には、実装もシンプルになるわけです。しかし、これはビジネスロジックが複雑になると実装が指数的に難しくなっ

注3）実は、ホストの電文もCOBOLのデータ型への割付けはやっていたのですけど。
注4）C#のstructと違って、IDLのstructにはメソッドもプロパティも定義できなかったのです。辛かった……。
注5）アプリケーションサーバーという名前に変わっただけという考え方もありますけど。