CHAPTER 1

Webサービスを使って3階層Windowsアプリケーションを作る

>>> C/Sシステムをどのように構築するか

大澤 文孝 OSAWA, Fumitaka

はじめに

3階層モデルの「3階層」とは、改めて言うまでもなく、「プレゼンテーション層」「ビジネスロジック層」「データストア層」の3つの層のことです。

3階層モデルでは、データ操作する処理をビジネスロジック層にまとめるため、それぞれのクライアントが直接データベースにアクセスする実装に比べて、データの整合性を保ちやすいのが特徴です。

また、処理がすべてビジネスロジック層に集約されますから、プレゼンテーション層の差し替えが容易です。

たとえば、WebアプリケーションとWindowsアプリケーションの両方をユーザーに提供しなければならない場面では、その共通ロジックをビジネスロジック層に仕込む3階層モデルを使って構築することになります。

.NET Frameworkで3階層モデルのアプリケーションを構築する場合、ビジネスロジック層を、どこに構成するのかには、次の選択肢があります。

COM+を使う

.NET Frameworkでコンポーネントを作成し、それをCOM+として登録して呼び出す方法です。

従来のWin32 API時代からあるもので、一般的な実装方法です。

COM+では、コンポーネント単位でのトランザクション機能もサポートされます。

.NET FrameworkのコンポーネントをCOM+として登録するには、厳密名で署名し、GAC（グローバルアセンブリキャッシュ）に登録することになります。

ストアドプロシージャとして構成する

データベースのストアドプロシージャとして構成する方法です。

従来のSQL Serverでは、ストアドプロシージャをT-SQLで書くしかなかったため、複雑な処理ができないとか、開発者がT-SQLの文法を理解しなければならないなどの問題がありました。

しかしSQL Server 2005では、VB.

レベル >>> Level

1 2 3 4 5

言語 >>> Language

- Visual Basic

ツール >>> Tool

- Visual Studio 2005 Professional
- SQL Server 2005 Express
- ASP.NET

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。

NETやC#を使ってストアドプロシージャを作れるようになったので、ビジネスロジックをストアドプロシージャとして実装する方法は、あながち的外れではなくなってきています。

とはいえ、ビジネスロジックは、データベース操作に限ったものではありませんから、すべてのビジネスロジックをストアドプロシージャとして実装するのは、データベースサーバーへの負荷やセキュリティの問題などを加味すれば、無理があります。

また、データベースシステムが決め打ちとなり、他のデータベースシステムに変更することも困難になります。

Webサービスとして構成する

IIS上に、Webサービスとして構成する方法です。

Webサービスへは、HTTPプロトコルでアクセスするため、配置したビジネスロジックは、Windows以外のOSからも使えるというメリットがあります。

反面、XMLデータの解析ロジックが必要になるため、COM+やストアドプロシージャとして構成する場合に比べて、パフォーマンス面で若干不利となります。

どの方法をとってもよいのですが、本稿では「Webサービスとしてビジネスロジックを構成する」という方法をとることにします。

そして作成したWebサービスを、Windowsアプリケーションから呼び出す方法を説明します[注1]。

注1）ここで提示した3種類の方法は、ひとつしかとれないわけではありません。アプリケーションによっては、いずれかを組み合わせて多段にすることもあります。とくにストアドプロシージャは、データベースに対して複雑な処理をする際に、COM+やWebサービスと、よく組み合わせられます。

銀行口座アプリケーション

本稿では、図1に示す銀行口座アプリケーションを作成します。

構築のポイントとなるのは、以下の3点です。

①大量データの受け渡し

すべての口座一覧は、後述するaccountテーブルに含まれています。

管理者向けの画面では、口座の一覧を取得できる必要があるでしょう。つまり、accountテーブルの内容を、クライアントへと返す必要があります。

このとき、まず、どのようなデータ構造でaccountテーブルを返すかという問題が浮上します。

この問題は、.NET Frameworkにおいては、DataSetオブジェクトを返すことで、容易に解決できます。なぜならDataSetオブジェクトは、Webサービスにおいて、XMLデータに変換して送受信することができるようになっているからです。

しかし問題は、それだけではありません。

現実的な実務アプリケーションにおいて、よく問題となるのが、大量データの送受信です。

転送時間やネットワークの帯域を考えた場合、たとえば10万もの口座数があるなら、そのすべてをそのつどクライアントに返すことは、現実的に難しいと言えるでしょう。

そこでクライアントが本当に必要とする範囲のデータだけを返すようにする必要があります。

図1に示したように、本稿で例示するアプリケーションでは、ユーザーにデータ一覧を表示するために、DataGridViewコントロールを使います。

そこで、「ユーザーがDataGridViewコントロールで見ている範囲」だけを返すようにすれば、効率が良さそうです。

しかしそれだと、ユーザーがスクロールするたびに、Webサービスへのアクセスが発生するため、操作性が悪くなります。

そこで、ある程度の操作性を向上させるために、ユーザーが見ている範囲の前後を少し多めに一括して取得し、キャッシュしたデータを表示するという方法をとることができます[注2]。

②更新処理

C/Sシステムは、複数のユーザーが同時に使います。

そのため、「あるユーザーが変更を加えようとしたときには、別のユーザーがすでに変更を加えていた」ということもありえます。

このような事態が起きたとき、無視して書き込むと、他人が書き換えたデータのさらなる上書きが発生してしまいます。

そこでデータが書き換わっていないことをチェックし、データが書き換わっているならば、エラーとして、ユーザーに再編集させるという処理が必要です。

この問題を解決するには、レコードの値が書き換わったときには値が変化するtimestamp型を使います。

注2）もっとも口座情報は、頻繁に変更される種の情報ではありません。そこで、残高など刻々と変化する列以外は、深夜のバッチ処理などでクライアントに転送するという方法もとれるでしょう。