Visual Studioで構築するエンタープライズシステム

Application Architecture for .NETの利用例

第5回 ユーザーインターフェイスを実装する（後編）

株式会社CSK
eソリューション技術部
中垣 健志
NAKAGAKI, Kenji

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。

システムの外見

梅雨もまだ明けていないというのに、都心部はうだるような暑さに見舞われています。お昼に連れ立って外に出ると、まるで服を着たままサウナに入っているかのような錯覚におそわれます。

汗を拭きながらふと隣を見ると、並びの席で仕事をしているIさんがシュルルっとネクタイを外しました。趣味のサーフィンで日に焼けた胸元をみて「お、クールビズ」とN君が思わず口走ると、Iさんは少し怒ったそぶりで「ネクタイ取ると、みんな僕のことクールビズっていうんですよね。僕はクールビズって名前じゃないぞって言いたい」と返しました。それはともかく、中身の人間性は変わらないのにネクタイをしているかしていないかという外見だけで人は他人を判断しがちです。中身も重要なのですが、外見もきっと同じくらい重要なのでしょう。「これはおそらくシステム開発にも当てはまるのではないか」、支援先でのお客さんとの画面デザインに関するやり取りを思い返しながらN君は思いました。

利用者からの要求

Soarアプリケーションが利用者に提供するインターフェイスは、Webアプリケーションです。利用者は、Internet Explorer、FireFox、OperaなどのPC上で動くブラウザ、そして場合によっては携帯端末などからアプリケーションを利用することになります。ここで一度、アーキテクトという役割から離れて利用者という観点でアプリケーションを考えてみてください。そして大手サイトのWebシステム、あるいは自社のイントラシステムを思い浮かべてみてください。システムに対する要求として出てくるものは何でしょうか？

「アプリケーションが3層構造になっていない」「データベース操作のSQL文が非効率的」といった内部の作り方に対する要求は通常出てこないでしょう。それよりも、「入力しにくい」「表示が遅い」「デザインのセンスがない」「プリンタが使えない」などといった表面に見える要求がよくあげられます。

利用者によるアプリケーションの評価は、目に見えるものがすべてです。どんなにビジネスロジックレイヤーやデータレイヤーの仕掛けがよく考えられていても、画面や操作方法が貧相であればそのアプリケーションは使えないという烙印を押されてしまうのです。したがって「利用者の使い勝手」という要求は、開発時の最重要課題として取り上げる必要があります。

◆ユーザーインターフェイスコンポーネントの役割

まずは先月号で紹介したプレゼンテーションレイヤーの内部構造をもう一度見てみます（図1）。

前回はこのうち、ユーザープロセスコンポーネントについて説明しました。そこで今回は、残るユーザーインターフェイスコンポーネントについて説明していきたいと思います。

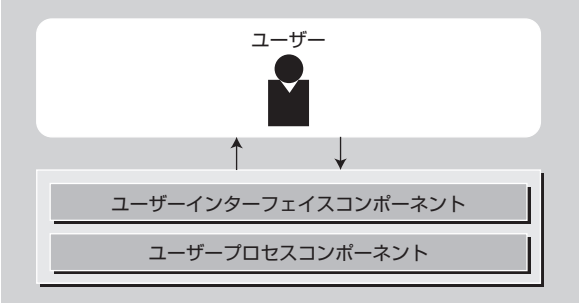
ユーザーインターフェイスコンポーネントが果たす主な役割は以下のとおりになります。

- ユーザープロセスコンポーネントが保持する情報を画面に表示する
- 画面に入力した情報をユーザープロセスコンポーネントに保存する

拍子抜けするほど少ないのですが、これがユーザーインターフェイスコンポーネントの果たすべき役割です。逆に言えば、ユーザーインターフェイスコンポーネントの役割をここまでそぎ落とすために、先月号まで紹介してきたデータアクセスクラスからユーザープロセスコンポーネントまでで、アプリケーションで必要なほとんどの処理を実装してきたのです。では、ここまで役割を少なくした理由は何だったのでしょうか？

利用者の画面に対する要求は、プロジェクトの初期から

図1：プレゼンテーションレイヤーの内部



リリース寸前まで（あるいはリリース後も）休むことなく発生することが多いものです。したがって基本設計や詳細設計で仕様を固めてしまい実装では設計書どおりに作るだけ、というわけにはなかなかいきません。かといって、利用者から要求があるたびにプログラムを修正して、作業が大きく手戻りしてしまうことは避けなければなりません。

そこで、ユーザーインターフェイスコンポーネントというカテゴリを用意して、画面に関する実装を行なう部分をすべてここに局所化してしまったのです。局所化することでユーザープロセス以下のクラスとは独立してこまめに修正ができるようになります。

◆コードビハインド

ここまでユーザーインターフェイスコンポーネントの役割について説明してきました。ASP.NETではこのユーザーインターフェイスコンポーネントが果たすべき機能を実装する場所として、「コードビハインド」という概念がすでに提供されています。

コントロールとユーザープロセスのデータ連結

ASP.NETでは「コントロールのデータ連結」という考え方があります。これは、コントロールにデータを連結すると、連結されたコントロールが自動的にデータを表示してくれるという考え方です。例としてよく挙げられるのはDataGridコントロールです。DataGridコントロールのDataSourceプロパティにDataSetやDataTableをデータ連結すると、DataGridコントロール自身がDataSetやDataTableからデータを取得して、画面に表示してく