WPF :Objets dynamiques bindé avec ADO.net

Auteur : Hassan KANDIL
Date de publication : 18/02/2013
Revu et adapté le : 22/12/2015

Il faut noter que la gestion des objets dynamiques est apparue avec la version 4.5, donc les applications adressées à des environnements avant 4.5 ne peuvent pas s'exécuter avec cette gestion d'objets dynamiques.

Il n'empêche qu'il est très utile d'utiliser cet avantage dans la gestion ADO.net.
Généralement, on utilise soit des services, soit des objets, soit des bibliothèques de classes pour accéder à notre base de données.

Le passage d'une catégorie à l'autre (Service/Object request/Bibliothèque) est une question d'architecture, et de choix de compilation. La programmation elle même aura toujours la même logique.
Concentrons nous sur une classe de lecture SQL avec une chaîne de connexion et une méthode de lecture de masse. Ceci nous impose de stocker les données dans une variable structurée, dont la structure est variable selon la requête initiatrice de la lecture.

Le premier pas est la construction d'une application WPF qui fait appel à une méthode de lecture ADO.net et qui affiche les informations retournées par la lecture.
La fenêtre principale décide de sa requête, La méthode de lecture n'a aucune idée de la structure des DATA retournée. Il est important de penser que la méthode de lecture qui doit se trouver dans une bibliothèque de classe est généraliste, donc elle est adressée à tout appelant. La seule propriété qui pourrait être commune à tous les objets communiqués à la méthode est Add pour ajouter les informations récupérées. Le binding lui même est placée dans le XAML et/ou le code behind.

Voici un exemple d'une classe de lecture qui reçoit une requête et qui doit retourner les données dans une structure variable.

L'application suivante essaiera d'installer les données dans une listview gérée dynamiquement. Elle doit définir les noms des colonnes en fonction de la requête et alimenter la listview par les données de la base, à travers un binding de préférence.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections.ObjectModel;
using System.Windows;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Controls;
using System.Windows.Data;

namespace FindDataSQL
{
    delegate void TrtExcept();
    class ReadData
    {
//BIGDAT est le nom du serveur de données Clients est le nom de la base .
        public static String connexionString = @"Data Source=BIGDATA;Initial Catalog=Clients;Integrated
Security=false;";

        #region Lecture dans une structure de taille variable
//La fonction reçoit la requête et la listview ou l'objet d 'affichage qui sera bindé aux datas
//La dataStruct est une observablecollection qui contiendra les données récoltées par la requête
```

```
        public void ReadInObject(string req, dynamic objReq, dynamic dataStruct)
        {
            IDbConnection connexion = new SqlConnection(connexionString);
            IDbCommand commande = connexion.CreateCommand();
            commande.CommandText = req;
            commande.CommandType = CommandType.Text;
//Permet de gérer les exceptions
            if (!LocalExcept(connexion.Open)) return;
            IDataReader lire = commande.ExecuteReader();
//Création des colonnes dans la listview selon les champs retournés
            CreatDynamic(objReq, lire);
            long j = lire.FieldCount;
            while (lire.Read())
            {
                string[] s1 = new string[j];
                for (int i = 0; i < j; i++)
                {
                    var v = lire.GetValue(i);
                    s1[i] = v.ToString();
                }
//On alimente la structure qui contient un nombre variable de colonnes
                dataStruct.Add(new MultiColumns { ValCol = s1 });
            }
            lire.Close();
            connexion.Close();
            connexion.Dispose();
        }
        #endregion
        public bool LocalExcept(TrtExcept fctExecute)
        {
            try
            {
                fctExecute();
            }
            catch (Exception e)
            {
                MessageBoxButton button = MessageBoxButton.YesNoCancel;
                MessageBoxResult result = MessageBox.Show(e.Message, "Attention", button);
                return false;
            }
            return true;
        }
//Cette méthode ne fait que créer les colonnes . l'ordre définit le lien en binding avec multiCol
//Autrement la colonne 0 est bindée au tableau string ValCol[0] et ainsi de suite
        public void CreatColumns(dynamic lvTest, IDataReader lire)
        {
            bool a;
            string s;
            int i = lire.FieldCount;
            for (int j = 0; j < i; j++)
            {
                if (j == 0) a = true;
                else a = false;
                s= "ValCol[" + j.ToString() + "]";
                MultiColumns.AddCol(lvTest, lire.GetName(j).ToString(), s, 100, a);
            }
        }
    }
}
```

out semble être basé sur la structure multicolumns qui en fait n'est qu'une simple classe qui définit le contenu d'une ou de plusieurs colonne (on peut ajouter name et type pour sophistiquer le traitement)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections.ObjectModel;
```

```
using System.ComponentModel;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;


namespace FindDataSQL
{
    public class MultiColumns : INotifyPropertyChanged
    {

        public static ObservableCollection<MultiColumns> multiCol = new
ObservableCollection<MultiColumns>();

        private string[] valCol;

        public string[] ValCol
        {
            get { return valCol; }
            set { valCol = value; NotifyPropertyChanged("ValCol");}
        }

        public event PropertyChangedEventHandler PropertyChanged;
        public void NotifyPropertyChanged(string propertyName)
        {
            if (PropertyChanged != null)
                this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }

//Le bind dynamique doit se faire avec ValCol[i] de l'objet multiCol
//la variable init permet de réinitialiser le nombre de colonnes de la listview
//true première colonne - false ajouter la colonne
        public static void AddCol(ListView lvActuel, string title, string bindElement, double width,
bool init)
        {
            GridView gv = (GridView)lvActuel.View;
            if (init == true) gv.Columns.Clear();
            gv.AllowsColumnReorder = true;
            gv.ColumnHeaderToolTip = "Dynamic Information";

            GridViewColumn Item = new GridViewColumn();
            Item.Header = title;
            Item.Width = width;

            Item.DisplayMemberBinding = new Binding(bindElement);

            gv.Columns.Add(Item);

            lvActuel.View = gv;
            lvActuel.ItemsSource = multiCol;
        }
    }
}
```

Il suffit d'alimenter la variable multicol pour que la listview affiche dynamiquement les données.
Pour cela on va ajouter une fenêtre de test qui accède aux données et qui les affichent dans la listeview.

```
<Window x:Class="FindDataSQL.WinExemple"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:FindDataSQL"
        mc:Ignorable="d"
```

```
        Title="WinExemple" Height="600" Width="800">
    <Grid>
        <Canvas Margin="0,0,-0.4,0">
            <ListView Name="lvTest" Height="200" Width="795">
                <ListView.View>
                    <GridView>
                        <GridViewColumn />
                        </GridViewColumn>
                    </GridView>
                </ListView.View>
            </ListView>
            <Button Content="Test" Name="btnTest" Click="TestClick" Height="20" Width="100"
Margin="10,242,183.6,10"/>
        </Canvas>
    </Grid>
</Window>
```

L'événement Click sera traité dans TestClick de la sorte

```
    private void TestClick(object sender, RoutedEventArgs e)
        {
            ReadData rd = new ReadData();
            rd.ReadInObject("select * from Analyse where nrep=0", lvTest, MultiColumns.multiCol);
        }
```

Cette listview pourrait recevoir toutes données de la base et les afficher en colonnes . Certainement il y a encore des choses à faire au niveau de formatage et de gestion de type. Mais comme exemple il permettra de séparer le traitement de la base et de la présentation. Dons il correspond à l'architecture 3 tier et à WPF MVVM.