Presentación de ADO.NET

ADO.NET es un conjunto de clases, interfaces, estructuras y enumeraciones que permiten el manejo de los datos. Los diferentes componentes de ADO.NET permiten separar el acceso a los datos de su manejo. ADO.NET facilita también la utilización del lenguaje XML, al permitir la conversión de datos relacionales al formato XML o la importación de datos a los formatos XML en un modelo relacional. Hay dos modos de funcionamiento disponibles en ADO.NET:

- el modo conectado;
- el modo no conectado.

1. Modo conectado

En un entorno conectado, la aplicación o el usuario está permanentemente conectado a la fuente de datos. Desde los principios de la informática, ha sido el único modo disponible. Este modo presenta algunas ventajas en su funcionamiento:

- Es facil de gestionar: la conexión se hace al principio de la aplicación y luego se corta al cierre de ésta.
- El acceso concurrente es más fácil de controlar: como todos los usuarios están conectados de forma permanente, es más fácil controlar cuál trabaja con los datos.
- Los datos están actualizados: siempre gracias a la conexión permanente a los datos, es fácil avisar a todas las aplicaciones que utilizan los datos de que se acaban de producir algunas modificaciones.

Por el contrario, ciertos inconvenientes vienen a oscurecer el panorama:

- Se debe mantener la conexión de red constantemente: en caso de utilización de la aplicación en un ordenador portátil, el acceso a la red se arriesga a no estar disponible permanentemente.
- Se corre el riesgo de infrautilizar los recursos del servidor: en el momento de establecer una conexión entre una aplicación cliente y un servidor, se reservan recursos del servidor para la gestión de esta conexión. Estos recursos siguen monopolizados por la conexión, incluso aunque ninguna información transite por ella.

Sin embargo, en ciertas situaciones, la utilización de un modo conectado es ineludible. Es el caso, por ejemplo, de las aplicaciones que realizan procesos en tiempo real.

2. Modo no conectado

Un modo no conectado significa que una aplicación o un usuario no está conectado constantemente a una fuente de datos. Las aplicaciones de Internet utilizan a menudo este modo de funcionamiento. Se abre la conexión a los datos, se obtienen los datos y luego se cierra la conexión. El usuario trabaja con los datos a partir de su navegador, y se reabre la conexión para la actualización de la fuente de datos o la obtención de otros datos. Los usuarios que trabajan en ordenadores portátiles también son los principales usuarios de entornos desconectados. Un médico, por ejemplo, puede cargar por la mañana los historiales de salud de los pacientes que va a visitar durante el día, luego, por la tarde, actualizar las modificaciones en la base de datos. Las ventajas de un entorno no conectado son las siguientes:

- Se utilizan las conexiones durante la duración más corta posible. De esta manera, un pequeño número de conexiones disponibles en un servidor bastan para muchos usuarios.
- Un entorno desconectado mejora la escalabilidad y las prestaciones de una aplicación al optimizar la disponibilidad de las conexiones.

Sin embargo, el entorno desconectado comporta algunos inconvenientes:

- Los datos disponibles en la aplicación no siempre están actualizados. Por ejemplo, en el caso de nuestro médico, si su secretaria añade resultados de análisis después de que él haya recuperado los historiales médicos de estos pacientes, no podrá disponer inmediatamente de la información.

- En ocaciones pueden surgir conflictos durante la actualización de la información en la base. Durante el desarrollo de la aplicación, se debe asumir este tipo de problemas. Hay diferentes planteamientos disponibles para la gestión de estos conflictos:

    - Autorizar la prevalencia de las actualizaciones más recientes, sobrescribiendo los datos ya presentes en la base.

    - Autorizar la prevalencia de las actualizaciones más antiguas rechazando las nuevas actualizaciones.

    - Prever código que permite al usuario elegir lo que desea hacer en caso de conflicto durante la actualización.

3. Arquitectura de ADO.NET

La meta de ADO.NET consiste en facilitar un conjunto de clases que permite el acceso a las bases de datos. Hay dos tipos de componentes disponibles:

- Los proveedores de datos específicos a un tipo de base de datos. Aseguran la comunicación con un tipo específico de base de datos y permiten el trabajo con los datos directamente en la base en modo conectado. Sin embargo, las posibilidades son limitadas, ya que sólo se dispone de un acceso en modo lectura.

- Las clases de manejo de los datos, independientes del tipo de base de datos, incluso utilizables sin base de datos, permiten el manejo local de los datos en la aplicación.

4. Los proveedores de datos

Los proveedores de datos sirven de pasarela entre una aplicación y una base de datos. Se utilizan para recuperar la información a partir de la base de datos y transferir los cambios efectuados en los datos por la aplicación hacia la base de datos. Hay cuatro proveedores de datos disponibles en el Framework.NET:

- el proveedor para SQL Server;

- el proveedor para OLE DB;

- el proveedor para ODBC;

- el proveedor para Oracle.

Todos proponen la implementación de cuatro clases, básicas, necesarias para el diálogo con la base de datos:

- La clase **Connection** permite establecer una conexión con el servidor de base de datos.

- La clase **Command** permite pedir la ejecución de una instrucción o de un conjunto de instrucciones de SQL a un servidor.

- La clase **DataReader** facilita un acceso a los datos sólo en modo lectura. Al igual que en el caso de los archivos, este acceso es sólo secuencial y, por lo tanto, el conjunto de datos es recorrido sólo una vez y de atrás adelante.

- La clase **DataAdapter** se utiliza para asegurar la transferencia de los datos hacia un sistema de caché local a la aplicación llamado `DataSet` y para actualizar la base de datos, en función de las modificaciones efectuadas localmente en el `DataSet`.

Hay otras clases que están especializadas en la gestión de las transacciones o el paso de parámetros a una instrucción SQL.

a. SQL Server

El proveedor de datos para SQL Server utiliza un protocolo nativo para dialogar con el servidor de base de datos. Además, como accede al servidor sin hacer uso de capas de software adicional (OLE DB u ODBC), consume muy pocos recursos. Se puede utilizar con SQL Server a partir de la versión 7. Todas las clases de este proveedor de datos están disponibles en el espacio de nombres **System.Data.SqlClient**. En este espacio de nombres, el nombre de cada clase viene prefijado por `Sql`. Así, la clase que permite conectarse a un servidor SQL Server se llama `SqlConnection`.

b. OLE DB

El proveedor OLE DB utiliza la capa de software OLE DB para comunicarse con el servidor de base de datos. Puede utilizar este proveedor para dialogar con una base de datos que no dispone de proveedores específicos, pero que cuenta con compatibilidad OLE DB. Con esta solución, el proveedor no contacta con el servidor directamente, sino que usa un driver OLE DB para comunicarse. Para que esta comunicación sea posible, el driver debe implementar algunas interfaces. Todas las clases están disponibles en el espacio de nombres **System.Data.OleDb**. Los nombres de clase de este espacio de nombres vienen prefijados con `OleDb`. Para poder funcionar correctamente, este proveedor exige la instalación de MDAC 2.6 en la máquina (*Microsoft Data Access Components*) o una versión posterior.

c. ODBC

El proveedor ODBC utiliza un driver ODBC nativo para comunicarse con el servidor de base de datos. Este proveedor utiliza un driver ODBC nativo para la comunicación. El principio es idéntico al utilizado para el proveedor OLE DB. Todas las clases están disponibles en el espacio de nombres **System.Data.Odbc**. Los nombres de clases vienen prefijados con `Odbc`. Para poder funcionar correctamente, este proveedor exige la instalación de MDAC 2.6 en la máquina (*Microsoft Data Access Components*) o una versión posterior.

d. ORACLE

El proveedor para Oracle permite la conexión a una fuente de datos Oracle. Las clases están localizadas en el espacio de nombres **System.Data.OracleClient** y utilizan Oracle como prefijo de nombre.

5. Buscar los proveedores disponibles

Para asegurar el buen funcionamiento de una aplicación que utiliza un acceso a los datos, los proveedores deben estar instalados en el puesto cliente. La clase `DbProviderFactories` propone el método compartido `GetFactoryClasses`, que permite enumerar los proveedores de datos disponibles en el puesto. El ejemplo de código siguiente muestra el nombre, la descripción y el espacio de nombres raíz de cada uno de los proveedores instalados en el puesto de trabajo.

```
public static void listaProveedores()
    {
        DataTable resultado;
        //recuperacion de la lista de los proveedores en una dataTable
        resultado = DbProviderFactories.GetFactoryClasses();
        //recorrido de las columnas de la dataTable y visualización del nombre
        foreach (DataColumn columna in resultado.Columns)
        {
            Console.Write(columna.ColumnName + "\t");
        }
        Console.WriteLine();
```

```
        // recorrido de la dataTable y visualización de cada fila
        foreach (DataRow linea in resultado.Rows)
        {
            // recorrido de cada fila y visualización de cada campo
            foreach (DataColumn columna in resultado.Columns)
            {
                Console.Write(linea[columna.ColumnName] + "\t");
            }
            Console.WriteLine();
        }
        Console.ReadLine();
    }
```

6. Compatibilidad del código

En función del proveedor utilizado, debe importar el espacio de nombres correspondiente para tener un acceso fácil a las clases del proveedor. Sin embargo, como las clases de cada uno de los proveedores no llevan el mismo nombre, su código será específico para un tipo de proveedor. Sin embargo, es posible escribir código prácticamente independiente del tipo de proveedor. Para ello, en vez de utilizar las clases específicas a cada uno de los proveedores, puede utilizar como tipo de datos las interfaces que implementan. La utilización de una clase específica sólo es indispensable para la creación de la conexión. Una vez creada la conexión, puede trabajar únicamente con interfaces. El siguiente ejemplo de código hace la lista del contenido de una tabla de una base SQL Server usando únicamente interfaces.

```
public static void accesParInterfaces()
    {
      IDbConnection ctn;
      ctn=new SqlConnection();
      ctn=new SqlConnection("Data Source=localhost;Initial Catalog=
Northwind;Integrated Security=True");
      IDbCommand cmd;
      cmd=ctn.CreateCommand();
      ctn.Open();
      cmd.CommandText="select * from products";
      IDataReader lector;
      lector=cmd.ExecuteReader();
      Console.WriteLine("lectura de los datos en una base SQL server");
      while (lector.Read())
      {
          Console.WriteLine("numero : {0} nombre producto: {1}",
lector.GetInt32(0),lector.GetString(1));
      }
    }
```

La ejecución de este código muestra el siguiente resultado:

```
lectura de los datos en una base SQL server
numero: 1 nombre producto: Chai
numero: 2 nombre producto: Chang
numero: 3 nombre producto: Aniseed Syrup
numero: 4 nombre producto: Chef Anton’s Cajun Seasoning
numero: 5 nombre producto: Chef Anton’s Gumbo Mix
numero: 6 nombre producto: Grandma’s Boysenberry Spread
numero: 7 nombre producto: Uncle Bob’s Organic Dried Pears
numero: 8 nombre producto: Northwoods Cranberry Sauce
```

```
numero: 9 nombre producto: Mishi Kobe Niku
numero: 10 nombre producto: Ikura
numero: 11 nombre producto: Queso Cabrales
numero: 12 nombre producto: Queso Manchego La Pastora
numero: 13 nombre producto: Konbu
numero: 14 nombre producto: Tofu
numero: 15 nombre producto: Genen Shouyu
numero: 16 nombre producto: Pavlova
```

Si esta aplicación debe migrar luego hacia otro tipo de base de datos, sólo hay que modificar la fila relativa a la conexión. Si ahora los datos están disponibles en una base de Access, la creación de la conexión toma la siguiente forma:

```
ctn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\\Documents and Settings\\tgroussard\\
Mis documentos\\libro c sharp 2008\\capítulo 8\\NWIND.mdb");
```

La ejecución del código así modificado genera, efectivamente, el mismo resultado:

```
lectura de los datos en una base SQL server
numero: 1 nombre producto: Chai
numero: 2 nombre producto: Chang
numero: 3 nombre producto: Aniseed Syrup
numero: 4 nombre producto: Chef Anton’s Cajun Seasoning
numero: 5 nombre producto: Chef Anton’s Gumbo Mix
numero: 6 nombre producto: Grandma’s Boysenberry Spread
numero: 7 nombre producto: Uncle Bob’s Organic Dried Pears
numero: 8 nombre producto: Northwoods Cranberry Sauce
numero: 9 nombre producto: Mishi Kobe Niku
numero: 10 nombre producto: Ikura
numero: 11 nombre producto: Queso Cabrales
numero: 12 nombre producto: Queso Manchego La Pastora
numero: 13 nombre producto: Konbu
numero: 14 nombre producto: Tofu
numero: 15 nombre producto: Genen Shouyu
numero: 16 nombre producto: Pavlova
```

Por el contrario, conviene ser prudente y no utilizar instrucciones SQL específicas a un tipo de base de datos particular. Para facilitar la corrección del código, es preferible agrupar todas las instrucciones SQL en forma de constantes de tipo cadena de caracteres al principio de cada módulo. Con esta técnica, no tendrá que buscar instrucciones SQL en mitad de centenas de filas de código de Visual C#. También conviene ser prudente durante la utilización de parámetros en una instrucción SQL. El proveedor para SQL Server utiliza parámetros con nombre; por lo tanto el orden de creación de los parámetros no tiene importancia. El proveedor para OLE DB utiliza la posición de los parámetros en la instrucción SQL para los reemplazos durante la ejecución. El orden de la creación de los parámetros es, pues, en este caso, capital para el funcionamiento correcto de la instrucción.