Guillermo "Guille" Som

Acceso a datos con ADO.NET 2.0

A estas alturas a poca gente hay que explicarle qué es ADO.NET, al menos a aquellos que están en este "mundillo" de la programación con tecnologías de Microsoft; y si nos atrevemos a preguntar, nos dirán que ADO.NET es la forma que tenemos de acceder a bases de datos usando .NET Framework. Pero como cada día se incorporan nuevos programadores deseosos de aprender a utilizar .NET, en esta sección de queremos explicar qué es y cómo utilizar esa forma de acceder a datos.

» De asistentes y otras ayudas asistidas

En esta serie de artículos sobre ADO.NET de la sección *dnm.inicio* que empezamos en este número vamos a hablar de espacios de nombres, de clases, de interfaces y de otros objetos, componentes y controles que serán los que utilicemos para poder acceder a los datos almacenados en una base de datos utilizando ADO.NET; pero debido a que el entorno de desarrollo que utilizaremos es Visual Studio 2005, y su nuevo IDE ha destacado por sus asistentes, a muchos seguro que les resultará más cómodo empezar a crear sus aplicaciones por medio de ellos, aunque no siempre nos harían falta para acceder a los datos.

En honor a la verdad, debo decir que en un principio mi intención era la de no usar los asistentes para enseñar cómo acceder a datos con ADO.NET, particularmente porque a mí personalmente no me gustan los asistentes, tampoco me gustan los *DataSet* tipados, no me gustan los controles enlazados a datos y, en general, no me gusta nada que tenga que ver con las automatizaciones sobre las que no pueda tener un control total -o casi- sobre el código generado.

En las presentaciones de Visual Studio 2005 y en otros textos se trata sobre los asistentes, particularmente sobre los asistentes para acceso a datos, así que no vamos a ser menos y también usaremos esos asistentes, aunque no dejaremos de un lado el código "puro y duro". Código que podemos usar para modificar y personalizar el generado por esos asistentes o simplemente para escribir una aplicación basada en ADO.NET desde cero. Todo dependerá de dos factores: el primero será de las ganas que tenga el lector de modificar el código generado automáticamente, y el segundo será de lo bien o mal que quien suscribe explique cómo hacer esas cosas de acceder a datos usando ADO.NET; así que habrá para todos los gustos, eso sí, siempre desde el punto de vista de quien escribe estas líneas.

Pero como esta sección se divide en dos partes: *Fundamentos* y *Taller*, empecemos viendo los fundamentos de ADO.NET y en artículos posteriores nos encargaremos de la parte más práctica del acceso a datos.

ADO.NET 2.0: Los conceptos básicos

La versión de ADO.NET que se incluye en el .NET Framework que acompaña a Visual Studio 2005 es la 2.0. A pesar de ser una nueva versión, los cambios que se han producido con respecto a la anterior no son tan drásticos como los que hubo entre la versión ADO basada en COM (o ActiveX) y la primera versión para la plataforma .NET.

ADO.NET se ha caracterizado desde un principio por usar otra filosofía de acceso a datos basado en lo que se conoce como *acceso a datos en modo desconectado*. En la versión anterior (ADO a secas) también se podía acceder a datos de esa forma desconectada, pero no era algo natural o intrínseco de la propia tecnología, mientras que en ADO.NET es la forma por defecto. La ventaja de trabajar en modo desconectado principalmente consiste en liberar de trabajo a los servidores de bases de datos, además de permitirnos tener más usuarios que puedan acceder a esos servidores sin necesidad de tener que pagar mayor número de licencias. En modo conectado, lo habitual es que cada vez que el usuario utiliza una aplicación de acceso a datos mantenga una conexión abierta con el servidor, modifique los datos, y cuando cierre la aplicación es cuando se desconecta del servidor. De esta forma, si necesitamos utilizar muchas conexiones a un mismo tiempo debemos tener las licencias oportunas. En el modo desconectado esto ha cambiado, ya que en realidad la conexión directa con el servidor de bases de datos solo la necesitamos en dos momentos concretos: cuando accedemos

Guillermo "Guille" Som es Microsoft MVP de Visual Basic desde 1997. Es redactor de dotNetManía, miembro de *Ineta Speakers Bureau Latin America*, mentor de *Solid Quality Learning* Iberoamérica y autor del libro *Manual Imprescindible de Visual Basic .NET*.
http://www.elguille.info

al servidor para traer los datos que vamos a usar y cuando queremos actualizar los cambios que hayamos realizado. Mientras manipulamos esos datos no necesitamos la conexión y la podemos tener cerrada; por tanto, damos la oportunidad a otros usuarios para que puedan usar esa conexión que dejamos libre, lo que equivale a que más usuarios puedan acceder al servidor sin necesidad de comprar más licencias. Un caso práctico es la versión gratuita de SQL Server 2000/2005 (conocida como MSDE para la versión 2000 o SQL Server Express para la versión 2005), que solo permite 5 conexiones a un mismo tiempo.

La filosofía del modo desconectado

¿Cómo funciona el acceso a datos en modo desconectado?

La forma en que trabaja ADO.NET para acceder a los datos cuando está en modo desconectado es siguiente: primero se conecta al servidor de bases de datos usando clases especializadas, a continuación asigna los datos que le hayamos indicado a unas clases que nos permiten manipularlos por medio de una copia realizada en nuestro equipo, finalmente actualiza los datos que hayamos modificado, de forma que se guarden, eliminen, etc. en la base de datos.

De estos tres pasos, el primero y el último son los que necesitan conectar con la base de datos, es decir, son los que deben saber cómo conectar con el servidor de bases de datos, traer los datos que le indiquemos, además de que deben conocer cómo guardar los cambios que realicemos. Por supuesto, esos pasos los daremos utilizando algunos métodos que expondrán ciertas clases, digamos, especializadas. Especializadas porque deben conocer bien al servidor de bases de datos para poder ejecutar todas esas acciones. De hecho, como veremos en un momento, según el tipo de bases de datos a la que queramos acceder podremos usar diferentes clases, algunas de ellas más genéricas que otras, pero en cualquier caso deben conocer cómo funciona el motor de bases de datos para poder realizar esas acciones.

Sin embargo, cuando manipulamos los datos que hemos obtenido de la base de datos, lo haremos utilizando clases no especializadas. No especializadas en el motor que manipula esos datos, pero especializadas en manipular datos. Especializadas en saber qué columnas estamos usando, de qué tipos de datos son esas columnas. Especializadas en saber cómo agregar una nueva fila de datos o cómo eliminar esos datos.

Las clases básicas de acceso a datos

Para realizar esos tres pasos de acceso a datos con ADO.NET utilizaremos principalmente dos clases. Por un lado tenemos la clase que nos permite indicar qué datos son los que queremos "traer" de la base de datos y a qué servidor nos queremos conectar. Todo esto lo haremos habitualmente usando una clase de tipo `DataAdapter`, como veremos en la siguiente sección existen clases especializadas para cada motor que maneja esas bases de datos, por ejemplo, para acceder a una base alojada en un servidor de SQL Server utilizaremos la clase `SqlDataAdapter`. Para usar esta clase, podemos indicar en el constructor de la misma varios parámetros; los más habituales serán la cadena de selección, que no es otra cosa que la típica cadena `SELECT` en la que indicaremos la tabla y los datos que queremos manipular y la cadena de conexión, en la que tenemos que especificar el servidor de la base de datos, el nombre de la base de datos en la que está la tabla a la que queremos acceder y la información de seguridad a usar.

> ADO.NET se ha caracterizado desde un principio por usar otra filosofía de acceso a datos basado en lo que se conoce como acceso a datos en modo desconectado

Una vez que tenemos creado el adaptador para acceder a la base de datos, le tenemos que indicar que traiga esos datos hasta nuestro equipo y que lo almacene en un objeto para manipularlo localmente y de forma desconectada. En el ejemplo que vamos a usar solo accederemos a una tabla, por tanto ese objeto puede ser uno del tipo `DataTable`, aunque también podría ser del tipo `DataSet`, que es el que se suele usar en muchos ejemplos de ADO.NET, pero como solo vamos a usar una tabla, no tiene mucho sentido complicar el ejemplo sobre todo porque en última instancia lo que usaríamos de ese `DataSet` sería la tabla que contiene la información, ya que un objeto `DataTable` sabe todo lo que hay que saber para manejar esos datos que queremos manipular, por tanto es más que suficiente para la mayoría de los casos.

En el fuente 1 mostramos el código necesario para acceder a una base de datos de SQL Server y cómo agregar esos datos a un control de tipo `ListView`. Veamos ese código y a continuación lo que hemos hecho para realizar las operaciones necesarias para traer esos datos y mostrarlos en el control `lstFilas` de tipo `ListView`.

Lo primero que hacemos es preparar dos variables en las que indicamos la cadena de conexión y los datos que queremos utilizar.

En la cadena de conexión (variable `cnnString`) indicamos que queremos usar la base de datos `Northwind` que está en el servidor local de SQL Server y para acceder a ese servidor vamos a usar la seguridad integrada de Windows, es decir, usaremos la identidad del usuario actual de Windows. Los datos que queremos traer los indicamos con una cadena de selección (variable `selString`) en la que le decimos que la tabla donde están los datos es `Employee` y los campos son los que están entre `SELECT` y `FROM`, aquí podemos usar un asterisco (*) para indicar todos los campos.

Creamos el adaptador al que pasamos como argumentos del constructor los datos

```
// La cadena de conexión, indicando el servidor, base de datos y la seguridad a usar.
string cnnString = "Data Source=(local); Initial Catalog=Northwind; Integrated Security=True";
// La cadena de selección indicando las columnas que vamos a usar.
string selString = "SELECT EmployeeID, FirstName, LastName, BirthDate, City, Country FROM Employees";
// Crear el objeto DataAdapter para acceder a la base de datos.
SqlDataAdapter da = new SqlDataAdapter(selString, cnnString);
// Crear la tabla que usaremos para manipular los datos.
DataTable dt = new DataTable();
// Obtener los datos indicados en la cadena de selección.
da.Fill(dt);
// Asignar los nombres de las columnas a la cabecera del ListView.
lvFilas.Columns.Clear();
foreach(DataColumn dc in dt.Columns)
{
    lvFilas.Columns.Add(dc.ColumnName, 120, HorizontalAlignment.Left);
}
// Asignar las filas al ListView
lvFilas.Items.Clear();
foreach(DataRow dr in dt.Rows)
{
    ListViewItem lvi = null;
    // Asignar el contenido de cada columna al ListView
    for(int i = 0; i < dt.Columns.Count; i++)
    {
        if(i == 0)
        {
            lvi = lvFilas.Items.Add(dr[i].ToString());
        }
        else
        {
            lvi.SubItems.Add(dr[i].ToString());
        }
    }
}
```

Fuente 1. Código de acceso básico a una base de SQL Server

de conexión y la cadena **SELECT**. Como el acceso lo haremos a una base de datos de SQL Server, utilizamos una clase especializada: **SqlDataAdapter**. El adaptador sabe todo lo que tiene que hacer para recuperar los datos que le hemos indicado en la cadena de selección, y mediante el método **Fill** le decimos que asigne esos datos en un objeto del tipo **DataTable**, que es el que posteriormente usamos para recuperar la información para mostrársela al usuario. La tarea del adaptador es conectarse a la base de datos y recuperar los datos; una vez que ya los ha asignado a la tabla que usaremos en memoria, cierra la conexión y se desliga del servidor. Como vemos, no necesitamos usar ningún otro objeto para realizar la conexión, solo el adaptador.

El objeto **DataTable** contiene una serie de propiedades en las que, entre otras, se incluyen las columnas (*Columns*) que hemos recuperado de la tabla indicada en la cadena **SELECT** además de las filas (*Rows*) con los datos correspondientes a cada una de esas columnas, y por medio de esas dos propiedades hemos asignado las columnas de la cabecera del **ListView** además de los elementos de ese control. En la figura 1 podemos ver el resultado de ejecutar el código del fuente 1.

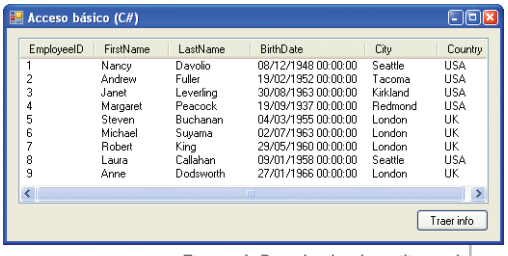

Figura 1. Resultado de utilizar el código del fuente 1.

Estos son los pasos que en principio debemos seguir. En los artículos siguientes veremos otros ejemplos prácticos en los que prepararemos el terreno para poder modificar esos datos, añadir nuevos y eliminar las filas que no necesitemos, pero básicamente los pasos que tendremos que dar serán los que acabamos de ver.

Ahora repasemos los diferentes espacios de nombres relacionados con las bases de datos que pone ADO.NET a nuestra disposición. También enumeraremos algunas de las clases que contienen esos espacios de nombres; algunos de ellos son exclusivos para la versión 2.0 de ADO.NET y por tanto solo disponibles desde Visual Studio 2005, tanto en las versiones normales (completas) como en las *Express Editions*.

Los espacios de nombres y las clases de acceso a datos

Todas las clases de .NET que están relacionadas con el acceso a datos están definidas en el espacio de nombres **System.Data**. Este espacio de nombres, además de incluir ciertas clases para manipular los datos (las más habituales las hemos usado en el ejemplo del fuente 1), a su vez contiene otros espacios de nombres con las clases especializadas en el acceso al motor de la base de datos a la que queremos acceder. Veamos cuáles son esos espacios de nombres y qué función tienen en toda esta "especialización" del acceso a datos con ADO.NET.

Según la función que proporcionan las clases podemos dividir en dos grupos los espacios de nombres de acceso a datos. Por un lado tenemos las clases que no están directamente relacionadas con los motores de acceso a datos, es decir, son independientes de si estamos trabajando con SQL Server, Oracle o cualquier otro proveedor de datos. En este grupo tenemos dos espacios de nombres: **System.Data** y **System.Data.Common**. Y por otro lado tenemos los espacios de nombres que contienen clases específicas de esos proveedores. En la versión 2.0 de ADO.NET se incluyen cinco espacios de nombres para otros tantos proveedores de datos. Esos espacios de nombres son:

- **System.Data.Odbc**, clases para acceder a datos cuyo origen es ODBC.
- **System.Data.OleDb**, clases para acceder a datos cuyo origen es OLE DB, la tecnología que la versión COM de ADO ha usado para acceder a datos, y será el proveedor que usaremos para acceder a bases de datos de Access.
- **System.Data.OracleClient**, clases específicas para acceder a bases de datos de Oracle.
- **System.Data.SqlClient**, clases específicas para acceder a bases de datos de SQL Server.

- **System.Data.SqlServerCe**, clases específicas para acceder a bases de datos de SQL Server Mobile, disponible solo en plataformas que utilizan Compact .NET Framework.

Todos estos espacios de nombres incluyen clases que están pensadas en el acceso a datos de tipo específico. No vamos a enumerar todas las clases (o las más comunes) ya que en realidad la funcionalidad es la misma la de las que se encuentran en el espacio de nombres **System.Data.Common**, el cual veremos en un momento, aunque los nombres usados para esas clases tienen un nombre acorde con el espacio de nombres en la que se incluye. Por ejemplo, en el fuente 1 hemos usado la clase **SqlDataAdapter** para comunicarnos con un servidor de SQL Server, si en lugar de acceder a ese tipo de servidor de bases de datos queremos conectar con un origen de datos por medio de las clases del espacio de nombres **OleDb**, el nombre de la clase equivalente es **OleDbDataAdapter**, y con el resto sería lo mismo. Cuando veamos las clases comunes indicaremos las clases equivalentes (en realidad las clases base) para cada uno de los cinco espacios de nombres que acabamos de enumerar.

Por último, veamos una pequeña descripción de los dos espacios de nombres con las clases que no dependen del proveedor de datos y las clases comunes en las que se basan las clases contenidas en los espacios de nombres de los proveedores de acceso a datos, así como algunas de las clases que contienen.

System.Data. Este espacio de nombres contiene las clases para manipulación de la información. Entre las clases más habituales contenidas en este espacio de nombres tenemos:

- **DataColumn**. Representa una columna de la base de datos, por medio de un objeto de este tipo podemos saber el tipo de datos de la columna, si acepta nulos, si es autoincremental, etc.
- **DataRow**. Representa una fila y por medio del indizador (propiedad predeterminada **Item** para VB) podemos acceder a cada uno de los datos de cada columna.
- **DataSet**. La estrella del acceso a datos en modo desconectado, puede contener información de varias tablas así como las relaciones entre ellas.
- **DataTable**. Representa la información de una tabla o vista almacenada en la base de datos. Contiene la información de las columnas y filas de los datos, además de las restricciones y claves primarias. Por medio de los métodos de esta clase podemos hacer búsquedas y selecciones sobre los datos que contiene la tabla, añadir nuevas filas, etc.
- **DataView**. Esta clase nos permite generar vistas basadas en una tabla, en la que podemos indicar los datos que queremos mostrar (incluso manipular).

System.Data.Common. En este espacio de nombres encontraremos las clases que comparten los proveedores específicos de acceso a datos. Éstas nos permitirán escribir código para acceder a los distintos proveedores, pero de una forma genérica, ya que muchas de las clases son clases abstractas; de esa forma podemos usarlas para crear nuestras propias clases de acceso a datos, sin necesidad de que dependan de un proveedor de datos en particular.

- **DataAdapter**. Esta clase será la que usaremos para acceder a la base de datos y asignar los datos a los objetos con los que los manipularemos de forma desconectada y finalmente volver a guardarlos.
- **DbDataAdapter**. Clase abstracta similar a **DataAdapter**. Esta es la clase base de las clases específicas de los proveedores de acceso a datos.
- **DbCommand**. Es una clase abstracta que representa un comando que puede ser una instrucción SQL o un procedimiento almacenado.
- **DbCommandBuilder**. Clase abstracta que se utiliza para generar los comandos usados para añadir, actualizar y eliminar datos de la base de datos. Estos comandos específicos los obtendremos por medio de métodos del tipo **Get[Acción]Command**, por ejemplo, para el comando **INSERT** sería **GetInsertCommand**.
- **DbConnection**. Clase abstracta utilizada para realizar una conexión a la base de datos. Los objetos **DbCommand** necesitan una referencia a un objeto de este tipo para poder ejecutar el comando correspondiente.
- **DbParameter**. Clase abstracta que representa un parámetro de un objeto **DbCommand**.
- **DbProviderFactory**. Clase abstracta que proporciona una serie de métodos para crear instancias de un proveedor de datos.
- **DbProviderFactories**. Contiene métodos estáticos que permiten crear instancias del tipo **DbProviderFactory** y para enumerar los proveedores de datos que tenemos instalados en el equipo.

Como comentábamos antes, los proveedores de datos específicos implementan estas clases, pero según el espacio de nombres del proveedor de datos, esas clases tendrán nombres diferentes. Por ejemplo, la clase abstracta **DbCommand** utilizará los nombres **OdbcCommand**, **OleDbCommand**, **OracleCommand**, **SqlCommand** o **SqlCeCommand** dependiendo del proveedor, lo mismo es aplicable al resto de las clases de **System.Data.Common** que empiezan con **Db**.

En próximos artículos veremos de forma práctica cómo usar estas clases para el acceso a datos que nos proporciona ADO.NET. También nos ocuparemos de los asistentes incluidos en Visual Studio 2005 que nos ayudarán a crear aplicaciones de acceso a datos prácticamente sin necesidad de escribir ni una sola línea de código. Aunque no nos olvidaremos de aquellos que prefieren controlar "al dedillo" todo el código usado para el acceso a datos; después será el lector el que decida qué usar para sus aplicaciones, incluso posiblemente sea una mezcla del código generado por los asistentes, pero personalizada para tener un mayor control de lo que ocurre tras el código autogenerado.

NOTA

ADO.NET incluye los espacios de nombres de los proveedores de bases de datos que hemos enumerado, pero otros proveedores de bases de datos pueden proporcionar sus propias librerías específicas para acceder a sus sistemas de bases de datos, tal es el caso de MySql. En cualquier caso, siempre podremos usar las clases de los espacios de nombres como OleDb para acceder a cualquier sistema de bases de datos, aunque es recomendable usar las clases especializadas porque estarán más optimizadas que las genéricas.