Nico De Greef

De teugels in handen met SMO

SQL SERVERBEHEER EN ONTWIKKELING GEAUTOMATISEERD

SQL-tools die net anders werken dan jij zou willen, behoren definitief tot het verleden. Met de SQL Server Management Objects (SMO) en Visual Studio kun je jouw eigen managementtools schrijven om jouw SQL Server-park op jouw manier te beheren.

De mogelijkheden om de kwaliteit van jouw databases en applicaties te verhogen is enorm. Durf je een overbodige view niet te deleten op SQL Server? Schrijf een SQL-dependency-checker die met één druk op de knop controleert of een view nog gebruikt wordt ergens binnen je SQL Server-park, zo op je websites, in rapporten of applicaties. Is jouw performance maar gemiddeld? Maak een performance-checker die controleert of alle tabellen wel op de juiste wijze geïndexeerd werden. De ingebouwde back-upfeatures voldoen niet? Schrijf dan een back-uptool volgens jouw normen. Ontwikkelaars kunnen een stored procedure of codegenerator schrijven op basis van tabelinformatie. Met de SMO-objecten kan heel wat gerealiseerd worden. Dit artikel is geen hands-on waar we van a tot z uitleggen hoe je een bepaalde applicatie kunt schrijven met SMO. We bespreken een aantal klassieke toepassingen. De code is beperkt tot het strikt noodzakelijke om het functionele aspect van SMO aan te tonen. De toegevoegde waarde zit hem in de waarschuwingen en tips. Hopelijk is met dit artikel jouw interesse voldoende geprikkeld om de applicaties volledig af te maken volgens jouw eigen wensen en zelf creatief aan de slag te gaan met SMO.

SMO en DMO

SQL SMO is een add-on voor het .NET Framework. Het bevat een heleboel classes die ons toelaten om alle facetten van Microsoft SQL Server programmatorisch te benaderen. Dat er boeken worden geschreven over SMO zal ons niet verbazen, tientallen classes staan ons ter beschikking om SQL Server tot in de kleinste details te beheren. Deze hulp-classes komen niet uit de lucht vallen sinds versie 2005 van SQL Server. Vorige versies van SQL Server beschikten al over de Distributed Management Objects (DMO), waarmee je via COM je SQL Server kon beheren. Sinds SQL Server 2005 spreken we niet meer over DMO, maar over SMO. SMO is niet het enige managementframework in SQL Server 2005. Bovendien beschikken we eveneens nog over frameworks voor het beheer van de Analysis Services (AMO) en de Replication Services (RMO).

Compatibiliteit

Betekent deze naamsverandering een breuk in de compatibiliteit? Nee, onze oude DMO-programma's kunnen nog steeds overweg met SQL Server 2005. Maar het benaderen van SQL Server 2005 gaat niet altijd even vlekkeloos. 'Could not connect to SQL server', zie afbeelding 1, is een foutmelding die wel eens verschijnt als je met je oude DMO-applicaties een SQL 2005 Server probeert te benaderen. Deze melding komt voor op pc's waar we nog niet langs geweest zijn met de SQL Server 2005 cd. Deze melding lijkt onheilspellend, maar is eenvoudig op te lossen. Installeer op de pc waar de DMO-applicatie draait eveneens de SQL-client connectivity-optie van de SQL Server 2005-installatie-cd. Aan de applicatie zelf zijn geen extra wijzigingen nodig. Na installatie van de client connectivity-optie kunnen oude DMO-applicaties eveneens met SQL 2005-servers verbinding maken, dit zonder herprogrammering. Functionaliteiten die nog niet bestonden ten tijde van SQL Server 2000 kunnen niet benaderd worden via DMO. Hiervoor moeten we overstappen naar SMO.

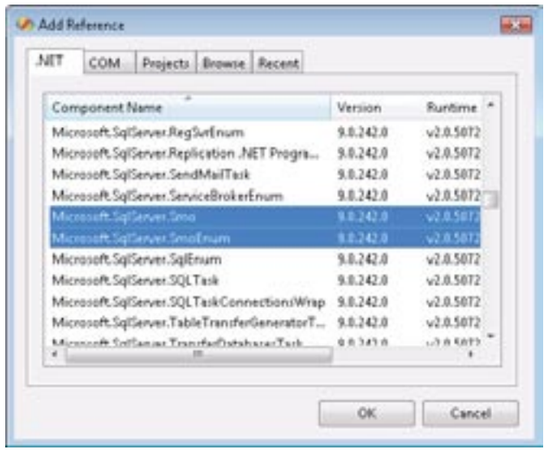
Afbeelding 1. Een foutmelding bij het benaderen van SQL Server 2005

SMO.NET installeren

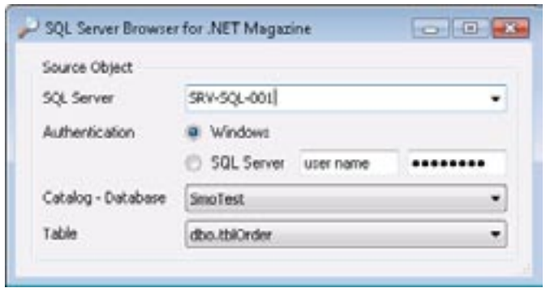
Om te kunnen ontwikkelen met SQL SMO hebben we allereerst de SMO dll's nodig. Deze bevinden zich op de SQL Server-installatiemedia. Deze worden geïnstalleerd als onderdeel van de client-componenten. Eenmaal geïnstalleerd kunnen we een nieuw project naar keuze aanmaken in onze vertrouwde Visual Studio-omgeving. In dit project leggen we referenties naar de SMO dll's, zoals in afbeelding 2. Meer hebben we niet nodig.

Serverobject

Het top-level object in het SMO-model is het serverobject. Dit vertegenwoordigt een SQL server. De andere objecten die tot een server behoren zijn childs van dit serverobject. De meeste SMO-applicaties werken dan ook volgens het zelfde principe. De eerste stap is het maken van een connectie met de SQL server door een serverobject te instantiëren. Eenmaal dit gedaan kunnen



Afbeelding 2. Referenties leggen naar SMO dll's



Afbeelding 3. Een voorbeeld van een simpele serverbrowser

we met alle methods en properties van deze Server-instance alle functionaliteiten van onze server beheren. Een volledig overzicht van het SMO-class-model kun je terugvinden in de referentielijst bij dit artikel.

Serverbrowser

In afbeelding 3 zie je een simpele serverbrowser die we gaan bouwen, een template die we in applicaties regelmatig kunnen hergebruiken. Als eerste taak gaan we de nodige referenties aanbrengen in ons nieuw .NET WinForms-project. Deze referenties zijn Microsoft.SqlServer.Smo en Microsoft.SqlServer.ConnectionInfo. Dit is een hele klus om steeds opnieuw te schrijven. We gaan ze in onze klassen dan ook stelselmatig vervangen door middel van een 'using' alias zoals in codevoorbeeld 1.

Om onze gebruikers het leven aangenamer te maken, gaan we hen een lijst tonen van de servers die beschikbaar zijn op het netwerk. Dit kan via de functie EnumAvailableSqlServers. In ons geval willen we op het hele netwerk zoeken. Hiervoor zetten we in de constructor de parameter bLocalOnly op false. Er bestaat ook een constructor om de instances op een specifieke host op te vragen. Deze functie geeft een DataTable-tabel terug waarin elke server een rij vertegenwoordigt. In tabel 1 zie je alle velden uit deze tabel. Het belangrijkste veld voor ons is het veld 'Name', dit bevat zowel host-name als instance-name. Meer info over het veld 'Version' kun je terugvinden op de Microsoft-site onder KB321185. De lijst met servers die deze functie teruggeeft, is niet limitatief, niet alle SQL-servers op het netwerk zijn er in aanwezig. Laat steeds aan de gebruiker de mogelijkheid open zelf een servernaam in te tikken.

Nu we weten welke SQL server van ons is, maken we er een verbinding mee. Er zijn twee manieren om verbinding te maken met SQL Server. Enerzijds hebben we de manier via Windows Authentication, anderzijds via SQL Authentication. De manier van inloggen bepaal je via een ServerConnection-object. In codevoorbeeld 2 zie je dat de Boolean LoginSecure-property

```
using smo = Microsoft.SqlServer.Management.Smo;
using smocmn = Microsoft.SqlServer.Management.Common;

/// <summary>
/// Fills the server combo
/// </summary>
private void InitServerCombo()
{
   cboServer.Items.Clear();
   DataTable dtServers;
   dtServers = smo.SmoApplication.EnumAvailableSqlServers(false);
   foreach (DataRow oRow in dtServers.Rows)
   {
      cboServer.Items.Add(oRow["Name"]);
   }
}
```

Codevoorbeeld 1. De serverlijst tonen

Field Name	Example
Name	LAPTOP001\\SQLEXPRESS
Server	LAPTOP001
Instance	SQLEXPRESS
IsClustered	false
Version	9.00.2047.00
IsLocal	true
Opmerking: Voor de standaard instance, is het Instance-veld gelijk aan de naam van de server	

Tabel 1. EnumAvailableSqlServers-info

bepaalt of je Windows Authentication (true) of SQL Authenticatication (false) gebruikt. In de meeste gevallen is dit voldoende. Het ServerConnection-object heeft echter nog vele andere properties die ingesteld kunnen worden. Je kunt onder meer het NetwerkProtocol kiezen om met de server verbinding te maken. Als je het ServerConnection-object hebt geconfigureerd, gebruik je het als parameter in de constructor van een Server-class. Nu we ons serverobject hebben geïnstantieerd, kunnen we pas echt van start gaan. Wat we in onze SQL-managementstudio (vroeger enterprise manager) kunnen terugvinden, is tevens een property van het serverobject. Om een specifiek column-object te krijgen, kun je Server.Databases[0].Tables[0].Columns[0] gebruiken.

In voorbeeldcode 3 zie je hoe we in de functie InitDatabaseCombo door de Server.Databases-collectie lopen. Deze collectie bevat objecten van het type database. Een databaseobject bevat een Boolean property IsSystemObject. Hiermee kun je de systeemdatabases (master, tempdb, ...) wegfilteren. De functie InitTableCombo werkt naar analogie met de InitDatabaseCombo. Idem dito voor views, stored procedures, enzovoort.

Back-uptool

Een toepassing van SMO is het maken van back-uptools. Een reden hiervoor kan zijn om de back-upwerkwijze in de reeds bestaande logging- en monitoring-strategie van een bedrijf te laten passen. Back-ups zijn meestal een langlopend proces. Wijzig daarom de int-property ConnectionContext.StatementTimeout van jouw server-instance. Deze bepaalt het aantal seconden alvorens de back-up stopt door een time-out. Codevoorbeeld 4 toont een eenvoudige back-uproutine. Het type back-up dat gemaakt wordt, is afhankelijk van twee properties van het back-upobject. De eerste property Action laat de keuze tussen Database, Log of File. Indien er voor het type databaseback-up wordt gekozen, laat de Boolean property Incremental de keuze

```
private smo.Server ConnectToServerFromUi()
{
   smocmn.ServerConnection oCon = new smocmn.ServerConnection();
   // Choose Server instance
   oCon.ServerInstance = cboServer.Text;
   // Choose authentication
   if (rbtnAuthtenticationWindows.Checked)
   {
      oCon.LoginSecure = true;
   }
   else
   {
      oCon.LoginSecure = false;
      oCon.Login = tbxUserName.Text;
      oCon.Password = tbxPassword.Text;
   }
   // Set Timeout
   oCon.StatementTimeout = 14400; // 4 hours
   // Connect
   return new smo.Server(oCon);
}
```

Codevoorbeeld 2. Connectie maken


```
/// <summary>
/// Fills the database combo based on the server combo
/// </summary>
private void InitDatabaseCombo()
{
   cboCatalog.Items.Clear();
   smo.Server oServer = ConnectToServerFromUi();

   // Enum databases
   foreach (smo.Database dataBase in oServer.Databases)
   {
      if (!dataBase.IsSystemObject)
      {
         // Only adds user databases
         cboCatalog.Items.Add(dataBase.Name);
      }
   }
}
```

Codevoorbeeld 3. Database enum

tussen een full (false) of incremental (true) back-up. De bron van de back-up wordt bepaald door de string-property Database. De doellocatie is moeilijker. SQL kan immers back-upen naar bijvoorbeeld files en tapes. Hiervoor moeten we een BackupDeviceItem (niet BackupDevice) toevoegen aan de Devices-collectie van ons object oBackup. File-locaties worden altijd bekeken vanuit het serverstandpunt! Een back-up naar C:\full.bak verwijst naar de C-drive van de server, ongeacht de pc waar de back-uptool draait. Een handig hulpmiddel hierbij is de Server.EnumDirectories(path)-functie. Deze functie geeft een DataTable terug met een lijst van de folders zoals op de server. Een Server.EnumDirectories(@"C:\")-implementatie zal zo alle rootfolders van de C-drive op de server weergeven. Voor security moet je er rekening mee houden dat de user die de back-ups maakt, niet alleen de nodige rechten heeft op de doellocatie, maar ook in staat is om back-ups te maken.
Restoren werkt in grote lijnen analoog aan het maken van back-ups. Zo zal het Restore-object ook werken met BackupDeviceItem-objecten. Interessante functies van dit object zijn de functie ReadBackupHeader die een datatable teruggeeft met alle mogelijke informatie over een back-upfile. Een interessante bron van informatie als je een tool wilt schrijven die back-upfiles inventariseert. Een andere functie is de Restore.RelocateFiles-collectie waar je RelocateFile-objecten aan toevoegt. Met een RelocateFile-object kun je logische databasefiles aan nieuwe fysische locaties koppelen. Ten slotte heb je hier de Boolean property NoRecovery conform de WITH (NO)RECOVERY-option van het respectieve sql-statement.

```
private void btnStartBackup_Click(object sender, EventArgs e)
{
   smo.Server oServer = ConnectToServerFromUi();
   smo.Backup oBackup = new smo.Backup();
   if (rbtnBackupFull.Checked)
   {
      oBackup.Action = smo.BackupActionType.Database;
      oBackup.Incremental = false;
   }
   if (rbtnBackupIncremental.Checked)
   {
      oBackup.Action = smo.BackupActionType.Database;
      oBackup.Incremental = true;
   }
   if (rbtnBackupLog.Checked)
   {
      oBackup.Action = smo.BackupActionType.Log;
   }
   oBackup.Database = cboCatalog.Text;
   oBackup.Devices.Add(new smo.BackupDeviceItem(tbxDestinationFile.Text_
   , smo.DeviceType.File));
   // Start Backup
   btnStart.Enabled = false;
   this.Cursor = Cursors.WaitCursor;
   oBackup.SqlBackup(oServer);
   this.Cursor = Cursors.Default;
   btnStart.Enabled = true;
   MessageBox.Show("De backup werd gemaakt.");
}
```

Codevoorbeeld 4. Back-up

Powershell

Visual Studio is niet de enige omgeving waarin je SMO kunt gebruiken. Dit kan evengoed in PowerShell-scripts. In de downloads zit een simpel PS-script om een tabel op het scherm te zetten met de databases en hun respectieve omvang van een gekozen server. Powershell is al aan bod gekomen in .Net Magazine #13. Het is niet de bedoeling om hier verder uitgebreid in te gaan op de PowerShell-syntax. Powershell kan een handige tool zijn om via de command-prompt kleine scripts te schrijven die er voor zorgen dat je als DB admin heel snel informatie kunt verzamelen tijdens troubleshoots.

Scripter

Ten slotte vermelden we nog het Scripter-object. Dit object genereert SQL-scripts van bestaande objecten in onze databases.

Nico De Greef is eigenaar van Denco Design (www.denco-design.be). Hij wordt hoofdzakelijk ingehuurd voor het bouwen van .NET-architecturen en het leiden van softwareprojecten. Hij is een MCSD.NET. Je kunt hem bereiken op ndg@denco-design.be .

Referenties

SMO Object Model: http://msdn2.microsoft.com/en-us/library/ms162209.aspx
KB321185: SQL Server version information: http://support.microsoft.com/kb/321185