Het ontsluiten van SharePoint Data Access

Wanneer gebruik je welke techniek en waarvoor?

Donald Hessing en Jan Steenbeek

Met de komst van SharePoint 2010 zijn er drie nieuwe data access technologieën geïntroduceerd voor het ontsluiten van SharePoint-objecten. LINQ to SharePoint ontsluit lijstdata aan de serverkant. Voor het ontsluiten van lijstdata aan de clientkant is de SharePoint REST interface, die net als LINQ to SharePoint met relationele data overweg kan. Tot slot is er het Client Object Model, waarmee je SharePoint objecten vanaf de client kunt benaderen.

De technieken hebben veel overlap met elkaar maar hebben zo hun eigen stijl. Maar wanneer gebruik je nou welke en waarvoor?

LINQ to SharePoint

Met SharePoint 2010 introduceert Microsoft een LINQ provider voor SharePoint waarmee het mogelijk is om lijstdata via LINQ (Language Integrated Query) te ontsluiten. De LINQ provider is beschikbaar in het Server Object Model (Server OM) van SharePoint 2010 en biedt de ontwikkelaar een alternatief voor het gebruik van Collaborative Application Markup Language(CAML). Met SharePoint Foundation wordt een nieuwe command line tool meegeleverd: SPMetal. Hiermee kunnen proxyclasses worden gegenereerd van de contenttypes en lijsten van een site. De gegenereerde code bevat classes voor elke lijst en voor ieder contenttype van de lijst. Een uitzondering hierop zijn de lijsten die gebaseerd zijn op een external contenttype. Deze lijsten worden niet door SPMetal gegenereerd en kunnen daardoor niet worden gebruikt in LINQ to SharePoint. Met enkele instellingen vanuit een XML configuratiebestand kan worden aangegeven voor welke lijsten en lijstinhoudstypes proxyclasses moeten worden gegenereerd. De SharePoint logica van contenttypes, die gebaseerd zijn op andere contenttypes, blijft behouden. Zo overerft de 'WikiPage' class de 'Document' class.

Doordat van de lijsten en contenttypes proxyclasses worden gegenereerd is het gebruik van LINQ to SharePoint strong-typed. Dit wil zeggen dat alle velden van een lijst beschikbaar zijn als property van de class en niet zoals bij het Client en Server OM als property bag (listitem["Title"]). De kracht van LINQ to SharePoint is dat voor de gegenereerde lijsten LINQ gebruikt kan worden voor het uitvragen van items uit een lijst. De LINQ provider bevat de logica om de LINQ query te vertalen naar een CAML query.
In voorbeeld 1 wordt het uitvragen via LINQ in de praktijk gebracht. Van een lijst met contacten worden alle verschillende bedrijven opgehaald.

```
using (LinqDemoDataContext _context = new LinqDemoDataContext
(SPContext.Current.Web.Url))
{
  IEnumerable<string> _bedrijven = (from p in _context.Contact-
  personen orderby p.Company select p.Company).
   Distinct();
}
```

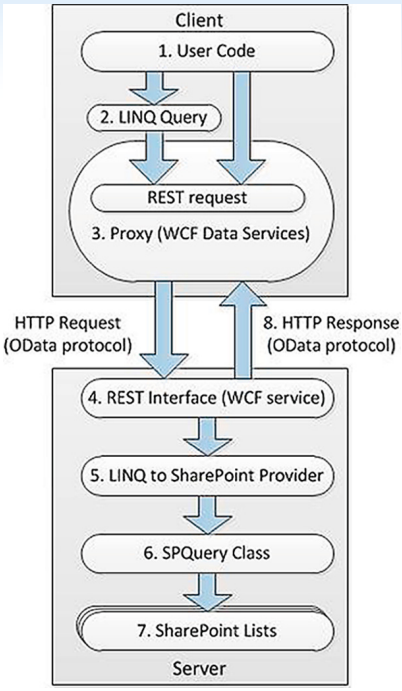
VOORBEELD 1: LINQ TO SHAREPOINT.

De code van voorbeeld 1 wordt door de LINQ to SharePoint provider omgezet in de volgende CAML query:

```
<View>
  <Query>
    <Where>
      <BeginsWith>
        <FieldRef Name="ContentTypeId" />
        <Value Type="ContentTypeId">0x010600BBB99F3118184C46821510D
        08A6264DC00</Value>
      </BeginsWith>
    </Where>
    <OrderBy Override="TRUE">
      <FieldRef Name="Company" />
    </OrderBy>
  </Query>
  <ViewFields>
    <FieldRef Name="Company" />
  </ViewFields>
  <RowLimit Paged="TRUE">2147483647</RowLimit>
</View>
```

VOORBEELD 2: CAML QUERY.

Merk op dat de CAML ook een controle op het contenttype bevat. Deze dwingt af dat items van het contenttype 'contactpersoon' zijn, of op 'contactpersoon' zijn gebaseerd. CAML bevat geen implementatie van het distinct keyword. Deze operatie zal daarom pas achteraf nadat alle data is opgehaald op de geretourneerde data worden uitgevoerd. Controleer daarom van iedere potentieel kostbare LINQ to SharePoint query of deze op een efficiënte manier naar CAML vertaald kan worden.



FIGUUR 1: SHAREPOINT REST COMMUNICATIE MODEL, MET DOOR VISUAL STUDIO GEGENEREERDE PROXY CLASSES VOOR DE CLIENT.

Voor- en nadelen

LINQ to SharePoint biedt voor een ontwikkelaar die geen kennis heeft van SharePoint, de mogelijkheid om op een voor hem bekende manier data te ontsluiten. De LINQ-implementatie retourneert een collectie van strong-typed objecten. Hierdoor kan tijdens compilatie van de code worden gecontroleerd of de properties van het item bestaan en of er geen ongeldige typecasts worden gebruikt. Doordat de LINQ-provider zelf verantwoordelijk is voor het vrijgeven van resource intensieve SharePoint-objecten zoals SPSite en SPWeb, hoeven ontwikkelaars hier zelf geen rekening mee te houden.
Een ander voordeel van LINQ to SharePoint is dat lijstdata vrij eenvoudig gecombineerd kan worden met gerelateerde lijstdata.

Een nadeel van LINQ to SharePoint is dat er een bug zit in de RTM versie van SharePoint Foundation. De bug zorgt ervoor dat bij gebruik in anonieme websites de gebruiker wordt gevraagd om zijn login gegevens. Dit is opgelost in de Cumulative Update (CU) van augustus en zal naar verwachting onderdeel worden van Service Pack 1. Zie hiervoor KB 2352342.
Een andere beperking van LINQ to SharePoint is dat de door SPMetal gegenereerde proxies alleen ondersteuning bieden voor fieldtypes die onderdeel zijn van SharePoint Foundation. Dit sluit bijvoorbeeld het gebruik van managed meta-data kolommen van SharePoint Server 2010 zonder additionele aanpassingen van de gegenereerde proxyclasses uit. De aanwezigheid van kolommen gebaseerd op deze fieldtypes zorgen er overigens niet voor dat deze lijsten niet werken in LINQ to SharePoint. Dergelijke kolommen komen in de gegenereerde classes gewoon niet voor. Een andere beperking is dat de SPMetal tool geen proxyclasses genereert voor externe lijsten die gebaseerd zijn op external contenttypes.

WCF Data Services

De LINQ-provider voor SharePoint biedt een krachtig middel om data op de server te ontsluiten. De SharePoint REST (Representational State Transfer) interface brengt deze kracht ook naar de client-kant. De REST interface is geïmplementeerd als WCF Data Services en is gebaseerd op de specificaties van het Open Data Protocol (OData). De OData specificatie is beschikbaar onder de Microsoft Open Specification Promise (OSP) en kan onafhankelijk van het platform en taal gebruikt worden. Zo zijn er bibliotheken beschikbaar voor onder andere Java, PHP en iPhone.

De SharePoint REST interface ontsluit lijstdata via een URL. Zo kunnen alle items van de documentbibliotheek opgevraagd worden door de URL http://portal/_vti_bin/listdata.svc/documents. Een uitzondering hierop zijn de lijsten die gebaseerd zijn op external contenttypes. Deze lijsten zijn niet via de SharePoint REST interface te raadplegen.

Met de SharePoint REST interface kan niet alleen lijstdata worden opgevraagd, maar kunnen ook items worden toegevoegd, bewerkt en worden verwijderd.

Voor het consumeren van de SharePoint REST interface aan de client-kant zijn diverse opties voorhanden. In figuur 1 wordt voor het consumeren van de SharePoint REST interface gebruik gemaakt van WCF Data Services. Door in Visual Studio een service referentie toe te voegen worden automatisch typed proxy classes gegenereerd. De LINQ provider van WCF Data Services vertaalt de LINQ query naar de REST query syntax (2). Vervolgens is de WCF Data Services implementatie verantwoordelijk voor het versturen van het request (3).

Aan de serverkant vertaalt de REST Interface het request eerst naar een LINQ query(4). De LINQ to SharePoint provider (5) vertaalt vervolgens de query naar een CAML query (6). De CAML query wordt uiteindelijke op de lijsten (7) losgelaten. De WCF service geeft standaard het bericht in Atom formaat terug (8). Dit lijkt in gebruik veel op RSS maar heeft een aantal extra opties, zoals het doorgeven van binaire data. Door de accept header van het request aan te passen kan informatie ook in JSON worden opgevraagd. Dit halveert ongeveer de hoeveelheid data die wordt uitgewisseld tussen client en server.

Query mogelijkheden

Een serviceaanroep bestaat uit de URL van de service met optionele query en filter parameters. Met de query parameter expand zijn eenvoudige joins van gerelateerde data mogelijk. De kolom die meegegeven wordt aan het expand keyword moet hiervoor een lookup kolom zijn naar de te joinen lijst

http://testportal/_vti_bin/ListData.svc	Geeft een overzicht van de diverse lijsten binnen de site terug.
http://testportal/_vti_bin/ListData.svc/lijst1	Geeft de inhoud van lijst 1 terug.
http://testportal/_vti_bin/ListData.svc/lijst1?$expand=CreatedBy	Geeft de inhoud van lijst 1 terug, met de volledige entries van de auteurs uit de userinformationlist

TABEL 1: VOORBEELD QUERY MOGELIJKHEDEN.

Via de URL kan ook een aantal filterparameters worden meegegeven. Zo kan een specifiek item uit de lijst worden opgevraagd of kunnen aan de hand van filterparameters bepaalde items geselecteerd worden:

http://testportal/_vti_bin/ListData.svc/lijst1(3)	Geeft het item met id 3 uit Lijst 1 terug.
http://testportal/_vti_bin/ListData.svc/lijst1?$filter=Bedrijf eq 'VXCompany'	Geeft alle items terug waar de waarde van de kolom bedrijf gelijk is aan 'VXCompany'
http://testportal/_vti_bin/ListData.svc/lijst1?$select=Bedrijf	Geeft alle items terug, maar van ieder item enkel de kolom Bedrijf.

TABEL 2: VOORBEELD FILTER MOGELIJKHEDEN.

In het derde voorbeeld van tabel 2 zorgt de select parameter er voor dat van alle items in de lijst alleen het veld bedrijf in het bericht wordt geretourneerd. Dit is belangrijk omdat standaard elke kolom van de lijst teruggegeven wordt, wat het bericht onnodig groot maakt. Zeker in de Atom notatie levert dit een aanzienlijk groter responsbericht op. Er zijn meerdere mogelijkheden om de REST interface te consumeren. De meest eenvoudige manier is de service als data source of service reference in een Visual Studio project op te nemen. Visual Studio zal dan op basis van de $metadata parameter van de listdata.svc service het schema van de data bron opvragen. Het schema beschrijft de beschikbare lijsten, de relaties en hun inhoudstypes. Aan de hand van deze beschrijving genereert Visual Studio de proxyclasses voor deze elementen. Dit zorgt ervoor dat de objecten net als bij LINQ to SharePoint strong-typed zijn. In voorbeeld 3 wordt van de eerste honderd documenten uit de "Shared Documents library" de titel en de auteur opgehaald. Door expliciet de properties Title, CreatedBy en UserName te selecteren in de query wordt er voorkomen dat alle properties door de client opgehaald worden.

```
PortalData.PortalDataContext _context;

_context = new PortalData.PortalDataContext(new Uri("http://test-
portal/_vti_bin/listdata.svc"));
_context.Credentials = CredentialCache.DefaultCredentials;

var _documenten = (from doc in _context.SharedDocuments
orderby doc.Title select new { doc.Title, doc.CreatedBy.
UserName }).Take(100);
```

VOORBEELD 3: WCF DATA SERVICES.

Voor en nadelen

Het aanbieden van databronnen op basis van OData wordt steeds populairder. Doordat OData gebaseerd is op een open standaard is het mogelijk om generieke client oplossingen te maken die voor verschillende databronnen geschikt zijn. Inmiddels zijn er voor diverse ontwikkelplatformen zoals Java en PHP bibliotheken ontwikkeld die OData databronnen zoals SharePoint, SQL Azure en IBM WebSphere kunnen consumeren.
Ontwikkelaars die gebruik maken van de WCF Data Services in combinatie met de door Visual Studio gegenereerde proxy classes, moeten rekening houden met het feit dat deze alleen ondersteuning biedt voor het Atom formaat. Dit is een beperking van de "System.Data.Services.Client.QueryResult" implementatie. Hierdoor wordt aanzienlijk meer dataverkeer tussen de client en de server verstuurd dan in de JSON notatie. De implementatie van de ASP.NET AJAX 4.0 voor WCF Dataservices maakt wel gebruik van de kleinere JSON notatie. Een andere beperking van de Dataservice implementatie is dat deze niet geschikt is voor anonieme websites.

Client Object Model

Het Client OM (Client Object Model) biedt ontwikkelaars de mogelijkheid om SharePoint Server Objecten aan te roepen vanaf een Client applicatie. Dit maakt het onder andere mogelijk om Rich Clients te bouwen bovenop het Microsoft SharePoint Framework. In voorbeeld 4 worden op basis van het Client OM alle lijsten opgehaald van de website die niet hidden zijn. Om gebruik te kunnen maken van het Client OM moet eerst de ClientContext gemaakt worden die onder andere de security context van de gebruiker bepaald. De constructor verwacht de URL van de site. Vervolgens worden met behulp van de Query syntax alle lijsten van de site die niet hidden zijn geselecteerd. De LoadQuery methode van de clientContext verwacht vervolgens een parameter van het IQuerable<T> type. Hierdoor kan gebruik gemaakt worden van de Query syntax zoals veel ontwikkelaars deze ook kennen van de SQL taal. De methode ExecuteQuery zorgt ervoor dat alle verzoeken van het Client OM worden verstuurd naar de server. Deze gebruikt hiervoor de WCF Service client.svc. De WCF Service client.svc zorgt er vervolgens voor dat alle properties van de geladen objecten worden verstuurd naar het Client OM. De Client OM runtime zorgt er op zijn beurt voor dat de Client OM objecten worden voorzien van de juiste waarden.

```
using (ClientContext clientContext = new ClientContext("http://
testportal"))
{
  var query = from list in clientContext.Web.Lists
        where list.Hidden != true
      select list;
  IEnumerable<List> listCollection = clientContext.
  LoadQuery(query);
  clientContext.ExecuteQuery();
}
```

VOORBEELD 4: LOADQUERY.

Load versus LoadQuery

In voorbeeld 5 wordt exact dezelfde query uitgevoerd maar nu op basis van de Load methode. De Load methode verwacht een query die is uitgedrukt als LAMBDA expressie. Hoewel de LAMBDA syntax voor veel ontwikkelaars lastiger is dan de Query syntax, is deze wel veel krachtiger. Veel query's voor het ophalen van data in het Client OM zullen gebruik maken van een LAMBDA expressie, al dan niet gecombineerd met de Query syntax.

```
using (ClientContext clientContext = new ClientContext("http://
testportal"))
{
  ListCollection lists = clientContext.Web.Lists;
  clientContext.Load(lists, collection => collection.
  Where(list=>list.Hidden!=false));
  clientContext.ExecuteQuery();
}
```

VOORBEELD 5: LOAD.

Een ander verschil is dat de methode Load het resultaat inlaadt in het meegeven object dat een referentie moet hebben naar een object van de ClientContext. De LoadQuery methode daarentegen geeft het resultaat in de vorm van een nieuw object van het type IEnumerable Collection terug.
Als gevolg hiervan kan de Load methode maar één keer worden aangeroepen voor het querien van bijvoorbeeld de ClientContext.Web.ListCollection, terwijl de methode LoadQuery aangeroepen kan worden voor het uitvoeren van meerdere queries op dezelfde ListCollection binnen een enkele roundtrip naar de server.

Optimaliseren van de query

In code voorbeeld 6 wordt de LoadQuery methode twee keer aangeroepen. In het eerste verzoek worden alle lijsten van het type 'Generic List' in de variabele genericLists geladen. In het tweede verzoek worden alle lijsten van het type 'Document Library' gela-

den in de variabele docLibraries. De ExecuteQuery methode zorgt ervoor dat deze in één roundtrip naar de server worden verstuurd. Iets wat met de Load methode niet mogelijk is omdat deze dezelfde ListCollection van het ClientOM refereert.

```
using (ClientContext clientContext = new ClientContext("http://
testportal"))
{

  Web site = clientContext.Web;

  IQueryable<List> query1 = from list in site.Lists
      where list.BaseType == BaseType.GenericList
                          select list;

  IQueryable<List> query2 = from list in site.Lists
      where list.BaseType == BaseType.DocumentLibrary
      select list;

  IEnumerable<List> genericLists = clientContext.LoadQuery(query1.
  Include(list => list.Title));

  IEnumerable<List> docLibraries = clientContext.LoadQuery(query2.
  Include(list => list.Title));

  clientContext.ExecuteQuery();
}
```

VOORBEELD 6: LOADQUERY IN BATCH.

Indien een object van het Client OM wordt geladen door de methode Load of LoadQuery wordt alleen een aantal standaard properties geladen. Dit zorgt ervoor dat de hoeveelheid dataverkeer al enigszins wordt beperkt. Het Client OM biedt echter ook de mogelijkheid dit verder te optimaliseren door alleen die velden te laden, die nodig zijn voor de applicatie. Dit wordt bereikt door gebruik te maken van de extension methode Include, die beschikbaar is op zowel de Query syntax als de LAMBDA expressie syntax. In voorbeeld 6 is een query gedefinieerd in de Query syntax die wordt uitgebreid met de LAMBDA expressie list =>list.Title. Dit zorgt ervoor dat alleen de property Title van de lijst geladen wordt. Het Client OM beperkt zich niet tot het ontsluiten van lijstdata, maar bevat een subset van het Server OM. Zo zijn er onder andere client classes voor ContentType, View, WebPart, Navigatie, Rollen en Workflow.

Intranet en Internet

Het Client OM is ook geschikt voor gebruik in publieke websites waar de gebruiker anoniem is. Voor het laden van bepaalde Client OM objecten zijn echter standaard een aantal beperkingen voor anonieme websites geconfigureerd. De beperkingen kunnen worden opgehaald door de ClientCallableSettings.AnonymousRestrictedTypes op te vragen van de betreffende WebApplication. Om de WebApplication te configureren voor het uitlezen van listitems zal de methode 'GetItems' van het type Microsoft.SharePoint.SPList verwijderd moeten worden van de AnonymousRestrictedTypes. Dit kan gedaan worden met onderstaand powershell script.

```
$wa.ClientCallableSettings.AnonymousRestrictedTypes.
Remove([Microsoft.SharePoint.SPList], "GetItems")
$wa.Update()
```

Client OM implementaties

Het Client OM is er in drie verschillende versies. Zo is er een Client OM implementatie voor .Net, Silverlight en JavaScript. Onafhankelijk van de implementatie moeten twee bibliotheekbestanden worden geladen om gebruik te kunnen maken van het Client OM. Voor de Managed .Net implementatie betekent dit dat de dll's van het Managed Client OM gedistribueerd moeten worden naar de clientcomputers. De Silverlight implementatie download de library als onderdeel van het XAP bestand terwijl voor JavaScript de bestanden sp.js en spcore.js op de webpagina geladen moeten worden.

Voor- en nadelen

Het Client OM bevat een aantal krachtige objecten die in het Server OM ook beschikbaar zijn. Het beperkt zich hiermee niet tot het ontsluiten van lijstdata. Doordat de verzoeken kunnen worden gebundeld kan de ontwikkelaar zelf bepalen wanneer en welke data wordt verstuurd tussen de client en de server. Het Client OM is daarnaast ook beschikbaar voor anonieme websites. De implementatie van het Client OM vereist dat de client en runtime bibliotheken geladen worden. Dit maakt de implementatie specifiek voor .Net, Silverlight of JavaScript clients. Het Client OM gebruikt voor de velden van een item een property bag. Dit maakt het Client OM weak-typed waardoor pas tijdens runtime gecontroleerd kan worden of een veld bestaat en of deze van het verwachte type is.

Samenvatting

De drie nieuwe data access technieken in SharePoint 2010 hebben ieder hun eigen karakteristieken. De LINQ to SharePoint en REST interface hebben veel overeenkomsten. Wanneer de REST interface als service referentie wordt toegevoegd aan een Visual Studio project zijn ze beide strong-typed, werken ze op basis van een LINQ query en maken joins op gerelateerde lijsten mogelijk. Het grote voordeel van de REST interface is dat deze gebaseerd is op een open standaard waardoor ook clients, gebaseerd op andere technologieën dan .NET, deze service eenvoudig kunnen consumeren.
Het Client OM is veruit de meest uitgebreide van de drie nieuwe data access technieken. Het is de enige van de drie technieken die op dit moment geschikt is voor externe lijsten. Daarnaast biedt het de ontwikkelaar de mogelijkheid om het aantal roundtrips en de hoeveelheid data die wordt verstuurd tussen de client en de server aanzienlijk te beperken. Het optimaal gebruik maken van het Client OM heeft echter wel de grootste leercurve van de besproken technieken.

Referenties

OData query en filter mogelijkheden
http://msdn.microsoft.com/en-us/library/cc907912.aspx.
MSDN: Working with the client object model
http://msdn.microsoft.com/en-us/library/ee857094.aspx#SP2010ClientOM_How_It_Works

Donald Hessing, is lead SharePoint architect bij VX Company en Microsoft Certified Master(MCM) voor SharePoint 2007. Hij blogt op: http://bloggingabout.net/blogs/donald. Email: dhessing@vxcompany.com

Jan Steenbeek, is sinds 2008 actief als SharePoint ontwikkelaar en consultant voor VX Company. Hij heeft zich met name gespecialiseerd in maatwerk voor intranet en extranet toepassingen die de functionaliteit van SharePoint uitbreiden. Email: jsteenbeek@vxcompany.com