Rob Lans is werkzaam bij The Competence Group.

Ongestructureerde data en het .NET Framework

EEN INFORMATIESYSTEEM ONTWIKKELEN VOOR ONGESTRUCTUREERDE DATA

Uit onderzoek blijkt dat meer dan 85% van alle informatie in een bedrijf bestaat uit ongestructureerde data. Bij ongestructureerde data ligt de opbouw van de informatie niet vast. De gebruiker bepaalt namelijk zelf de samenstelling en volgorde van de gegevens. Juist voor het ontwikkelen van softwareoplossingen voor dit soort informatie ontbreken de juiste tools en technologieën. In dit artikel komt het beheer van ongestructureerde data aan de orde.

Voorbeelden van ongestructureerde data zijn tekstuele documenten, spreadsheets, presentaties en webpagina's. Dit soort informatie wordt traditioneel buiten een relationele database opgeslagen. De reden hiervoor is dat de opbouw en structuur van dit soort informatie niet in een relationeel datamodel zijn vast te leggen. Dit wil niet zeggen dat er totaal geen structuur aanwezig is. Laten we als voorbeeld een tekstdocument in de vorm van een elektronisch boek nemen. Een boek is onderverdeeld in hoofdstukken, hoofdstukken zijn weer opgebouwd uit paragrafen, paragrafen bestaan uit woorden en woorden weer uit letters. Wat we hieruit kunnen concluderen is dat een boek bestaat uit een verzameling informatieobjecten die hiërarchisch gerangschikt zijn. Deze informatiestructuur is ook toepasbaar op de andere hiervoor genoemde voorbeelden. Wat we dus nodig hebben om dit soort 'ongestructureerde' data op te slaan is een systeem waarin we willekeurige informatieobjecten in een hiërarchische structuur kunnen opslaan.

Implementatie

We beginnen met het definiëren van de basis-class voor een informatieobject; zie codevoorbeeld 1. We noemen deze basis-class een InformationObject. Een InformationObject heeft een Naam en een ID. De Naam is de zichtbare naam van het object voor een gebruiker. Met de ID wordt het object op een unieke manier geïdentificeerd. Van deze basis-class worden dan de applicatiespecifieke informatie-elementen afgeleid.
Voor de opslag van de informatieobjecten naar een persistent medium gebruiken we XML. We moeten dus een manier bedenken waarmee we onze objecten eenvoudig in XML kunnen omzetten en omgekeerd. Gelukkig biedt het .NET Framework hiervoor uitkomst in de vorm van XmlSerialisatie. Met XmlSerialisatie kunnen we op eenvoudige wijze complete objecten en objectstructuren omzetten van en naar XML. Codevoorbeeld 2 toont hoe we XmlSerialisatie toepassen.

XmlSerialisatie

We maken een nieuwe instantie van de XmlSerializer-class en geven in de constructor het type van het object mee dat we willen omzetten naar XML. Voor het schrijven van XML roepen we de Serialize-methode aan en geven de stream mee waarin het XML-resultaat geschreven moet worden. Het lezen van het XML-bestand gaat op nagenoeg gelijke wijze. We maken weer een instantie van de XmlSerializer-class, hierna roepen we de Deserialize-methode aan en geven de stream mee waaruit gelezen moet worden. De Deserialize-methode levert als resultaat het object op. Na het aan-

```
using System;

namespace InformationStore
{
  public class InformationObject
  {
    public string Name;
    public Guid ID = Guid.NewGuid();
  }
}
```

Codevoorbeeld 1.

```
public class ObjectWriter
{
  public static void Write(object o, Stream stream)
  {
    XmlSerializer serializer = new XmlSerializer(o.GetType());
    serializer.Serialize(stream, o);
  }

  public static void Write(object o, string fileName)
  {
    using(Stream stream = File.Create(fileName))
    {
      Write(o, stream);
    }
  }
}

public class ObjectReader
{
  public static object Read(Type type, Stream stream)
  {
    XmlSerializer serializer = new XmlSerializer(type);
    return serializer.Deserialize(stream);
  }

  public static object Read(Type type, string fileName)
  {
    using(Stream stream = File.OpenRead(fileName))
    {
      return Read(type, stream);
    }
  }
}
```

Codevoorbeeld 2.

```
public class Image: InformationObject
{
   public int Width;
   public int Height;
   public string MimeType;
   public byte[] ImageData;
}

static void Main()
{
   Image image = new Image();
   image.Name = "Image";
   image.Width = 100;
   image.Height = 200;
   image.MimeType = "images/gif";
   image.ImageData = new byte[16];   // dummy data

   ObjectWriter.Write(image, @"image.xml");
}
```

Resultaat

```
<?xml version="1.0"?>
<Image xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <Name>Image</Name>
   <ID>3570a30a-adbf-4a74-bd79-c35c45acc637</ID>
   <Width>100</Width>
   <Height>200</Height>
   <MimeType>images/gif</MimeType>
   <ImageData>AAAAAAAAAAAAAAAAAAAAAA==</ImageData>
</Image>
```

Codevoorbeeld 3.

roepen van de ObjectReader.Read-methode moeten we daarna nog het resultaat casten naar het juiste type.
In codevoorbeeld 3 zijn een voorbeeld en het resultaat van het gegenereerde XML-document te zien. Wat opvalt is dat de Serializer automatisch voor alle velden een XmlElement aanmaakt met de naam die gelijk is aan de veldnaam. Dit is niet altijd wenselijk. We kunnen dit proces beïnvloeden door het plaatsen van een aantal attributen. Door een *[XmlElement("naam")]* attribuut te plaatsen op een veld kunnen we de naam van het gegenereerde element beïnvloeden. Door een *[XmlAttribute("naam")]* te plaatsen op een veld kunnen we ervoor zorgen dat het veld als attribuut verschijnt in plaats van een element.
Hoe werkt XmlSerialisatie nu precies? Bij de aanroep van de constructor van de XmlSerializer-class genereert het .NET Framework C#-code. De zeer efficiënte C#-code wordt gecompileerd tot een tijdelijke assembly en geladen in het applicatiedomein. De generatie- en compilatiestap wordt één keer gedaan voor elke Type dat meegegeven wordt aan de constructor van de XmlSerializer. Dit betekent dat bij de eerste aanroep van de constructor van de XmlSerializer-class voor een bepaald type enige vertraging optreedt.

SQL Server

In ons voorbeeld hebben we de informatieobjecten weggeschreven naar een XML-bestand. Voor een schaalbare oplossing willen we de informatieobjecten opslaan in een SQL Server-database. Deze database wordt dan ook gebruikt voor het vasthouden van de hiërarchische structuur van de objecten. We kunnen volstaan met het definiëren van één tabel. Afbeelding 1 laat de tabeldefinitie zien. De kolommen ObjectID en ParentID worden gebruikt voor de hiërarchische structuur. In de kolom TypeName schrijven we het type van het object zodat we bij het laden weer een instantie van het juiste type kunnen maken. De XmlData-kolom ten slotte wordt gebruikt om de XML van het geserialiseerde object in op te slaan.

```
public ArrayList Select(Guid parentID)
{
   ArrayList objects = new ArrayList();

   using(SqlConnection connection = new SqlConnection(ConnectionString))
   using(SqlCommand command = new SqlCommand("SELECT * FROM InformationObjects
      WHERE ParentID=@ParentID", connection))
   {
      command.Parameters.Add("@ParentID", SqlDbType.UniqueIdentifier).
         Value = parentID;
      connection.Open();
      try
      {
         using(SqlDataReader reader = command.ExecuteReader())
         {
            while(reader.Read())
            {
               Type type = Type.GetType(reader["TypeName"].ToString());
               byte[] bytes = (byte[])reader["XmlData"];
               Object o = ObjectReader.Read(type, bytes);
               objects.Add(o);
            }
         }
      }
      finally
      {
         connection.Close();
      }
   }
   return objects;
}

private string GetTypeName(object o)
{
   string[] splitName = o.GetType().AssemblyQualifiedName.Split(',');
   return string.Join(",", splitName, 0, 2);
}

public void Insert(Guid parentID, InformationObject o)
{
   using(SqlConnection connection = new SqlConnection(ConnectionString))
   using(SqlCommand command = new SqlCommand(@"INSERT INTO
      InformationObjects (ObjectID, ParentID, TypeName, XmlData)
      VALUES (@ObjectID, @ParentID, @TypeName, @XmlData)", connection))
   {
      connection.Open();
      try
      {
         string typeName = GetTypeName(o);
         byte[] bytes = ObjectWriter.Write(o);
         command.Parameters.Add("@ObjectID",
            SqlDbType.UniqueIdentifier).Value = o.ID;
         command.Parameters.Add("@ParentID",
            SqlDbType.UniqueIdentifier).Value = parentID;
         command.Parameters.Add("@TypeName",
            SqlDbType.VarChar).Value = typeName;
         command.Parameters.Add("@XmlData",
            SqlDbType.Image).Value = bytes;
         command.ExecuteNonQuery();
      }
      finally
      {
         connection.Close();
      }
   }
}
```

Codevoorbeeld 4.

InformationObjects

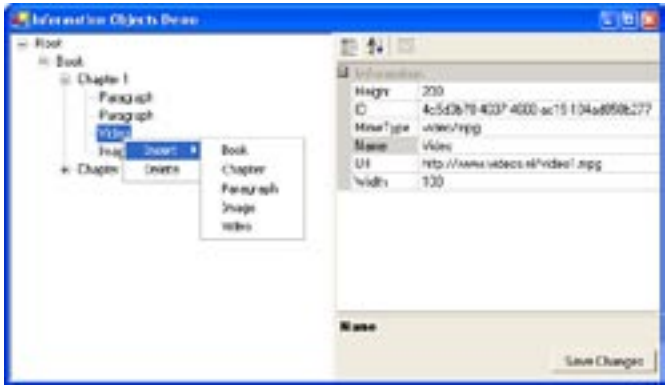
	Column Name	Data Type	Length	Allow Nulls
🔑	ObjectID	uniqueidentifier	16	
	ParentID	uniqueidentifier	16	
	TypeName	varchar	255	
	XmlData	image	16	

Afbeelding 1. Tabelstructuur

Codevoorbeeld 4 toont de implementatie voor het schrijven en lezen van een informatieobject van en naar een database. Door het wegschrijven van de naam van het type van het object zijn we in staat om bij het lezen, door middel van de aanroep *Type.GetType()*, een instantie van het object te maken van het juiste type. Voor het omzetten van het type naar een string maken we gebruik van de property *AssemblyQualifiedName*. Deze property geeft de volledige gekwalificeerde naam van het type in de vorm "Type, Assembly, Version, Culture, PublicKeyToken". We gebruiken de hulpmethode *GetTypeName* om alleen het "Type, Assembly"-gedeelte te krijgen. Hiermee wordt voorkomen dat bij wijziging van het versienummer, bijvoorbeeld door een rebuild in Visual Studio, het juiste type niet meer kan worden gevonden.

Demoapplicatie

De volledige implementatie is uitgewerkt tot een demoapplicatie; zie afbeelding 2. De applicatie bestaat uit een TreeView-control waarmee de hiërarchie van informatieobjecten zichtbaar gemaakt wordt. Daarnaast wordt een PropertyGrid-control gebruikt om de eigenschappen van het geselecteerde informatieobject te tonen.



Afbeelding 2. Gebruikersinterface van demoapplicatie

Ten slotte

In dit artikel is een oplossing beschreven voor het beheren van ongestructureerde data. De oplossing bestaat uit de opdeling van de informatie in kleine objecten, de zogenaamde informatieobjecten. De informatieobjecten krijgen structuur door ze hiërarchisch te rangschikken. Om tot een volledige implementatie voor een informatiesysteem voor ongestructureerde data te komen, is er nog een aantal openstaande punten zoals zoeken, versiebeheer en relaties. Deze punten zijn in een inmiddels gerealiseerd informatiesysteem opgelost, maar vallen buiten het bereik van dit artikel.

Rob Lans is werkzaam bij The Competence Group, www.competence.biz. Zijn e-mailadres is rlans@competence.biz

Nuttige internetadressen

http://msdn.microsoft.com/xml

De source-code voor de demoapplicatie is te vinden op: www.microsoft.nl/netmagazine7