Mark Willems en Dennis Mulder zijn werkzaam bij Avanade

Enterprise Library voor .NET 2.0

DE VERNIEUWINGEN OP EEN RIJ

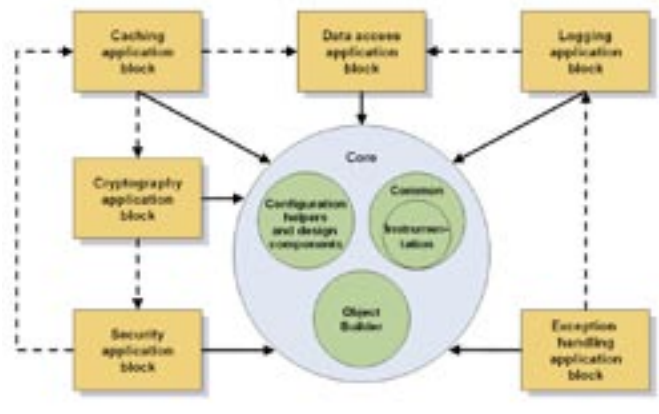
Het jaar 2006 is goed begonnen met de nieuwe release van Enterprise Library, vaak afgekort als EntLib voor het .NET Framework 2.0. In januari 2005 bracht de Microsoft Patterns & Practices Group ons de eerste release van EntLib. Met meer dan 200.000 downloads kan gerust gezegd worden dat Enterprise Library een succes is. Met dit artikel willen wij een overzicht geven van de vernieuwingen in Enterprise Library voor .NET 2.0.

Enterprise Library is ontstaan uit het integreren van zeven application blocks, te weten Data Access, Exception Handling, Configuration, Caching, Logging & Instrumentation, Security en Cryptography. Een application block kan gezien worden als een stuk software dat je voor veelvoorkomende ontwikkelscenario's kunt hergebruiken en uitbreiden. Zoals gezegd beleefden we de eerste release van EntLib in januari 2005 en volgde een update in juni 2005. Met de komst van het .NET Framework versie 2.0 in november 2005 heeft de Patterns & Practices Group besloten EntLib te verbeteren door optimaal gebruik te maken van de nieuwe functionaliteiten in de nieuwe framework-versie. Bij het schrijven van dit artikel is de definitieve versie van EntLib voor .NET 2.0 nog niet vrijgegeven en daarom is dit artikel gebaseerd op de 'interim community drop' van december 2005.

Architectuur

Deze nieuwe versie van Enterprise Library is ontworpen om optimaal gebruik te maken van de nieuwe mogelijkheden die het .NET Framework 2.0 ons biedt. Dit houdt in dat sommige onderdelen die we nog wel in eerdere versies van EntLib zagen nu zijn verwijderd vanwege de platformondersteuning. De API is echter nagenoeg identiek gebleven en de meeste veranderingen bevinden zich dan ook onder de motorkap. Op een aantal van die veranderingen gaan wij in de loop van dit artikel nog wat verder in. Voordat we dieper inzoomen op een aantal van de blocks in EntLib, bespreken we eerst het nieuwe architectuurplaatje zoals dat in afbeelding 1 is te zien.

We zien in het architectuurplaatje dat de individuele blocks zijn opgebouwd rondom de Core-functionaliteit van EntLib. We zien



Afbeelding 1. EntLib 2.0-architectuur

dat de Core-functionaliteit grofweg is opgebouwd uit drie onderdelen: een common assembly voor met name instrumentation, een configuratieklassenbibliotheek met wat design-time componenten en de configuratieconsole en als laatste de object builder waarover later meer. De Instrumentation-features zijn nu ook beter te configureren en zijn beter gedocumenteerd. Dit was vooral een probleem in versie 1.0 dat heel netjes door Patterns & Practices is opgelost. De individuele blocks zijn ontworpen met minimale afhankelijkheid, zodat ze individueel kunnen worden gebruikt maar ook in samenhang met elkaar.

Data-access

Omdat het data-accessblock één van de meest gebruikte blocks is, heeft het ontwikkelteam geprobeerd de API zoveel mogelijk hetzelfde te houden. Er zijn slechts wat kleine wijzigingen gedaan vanwege verbeteringen in ADO.NET 2.0. Zo zul je bij een migratie merken dat de DBCommandWrapper-klasse is vervangen door de DBCommand-klasse uit de System.Data.Common-namespace. Voor de connectionstring hebben we meer flexibiliteit gekregen. We kunnen ervoor kiezen de connectionstring uit de <connectionStrings>-sectie te gebruiken of, wanneer we geen configuratiefiles gebruiken, direct de connectionstring mee te geven in code. Een voorbeeld van dit laatste is te zien in codevoorbeeld 1. Voor de eenvoud hebben we hier de exception handling-code weggelaten. Voor het data-access block kunnen we nu gebruik maken van elke managed ADO.NET 2.0-provider. In de oude versie moesten we een wrapper schrijven om bijvoorbeeld gebruik te maken van een managed MySql-provider. In de nieuwe versie is een nieuwe klasse geïntroduceerd genaamd GenericDatabase, die ons nog meer flexibiliteit biedt bij het eventuele poorten van een applicatie naar een andere database. Deze klasse werkt niet alleen met elke provider, maar ook met OLE DB en ODBC.

```
public DataSet GetPlayersByTeam(string connectionString, int team)
{
   //Instantieer het Database object, met de specifieke connectionstring
   SqlDatabase db = new SqlDatabase(connectionString);

   //Voer de stored procedure uit met 1 regel code!
   return db.ExecuteDataSet("GetTeamPlayers", team);

   //Let op: de database connection wordt gesloten door
   //het aanroepen van de ExecuteDataSet methode
}
```

Codevoorbeeld 1.

```
LogEntry logEntry = new LogEntry();
logEntry.EventId = 100;
logEntry.Priority = 1;
logEntry.Message = "Debug informatie";
logEntry.Categories.Add("Debug");
logEntry.Categories.Add("Informatie");
Logger.Write(logEntry);

//Of een kortere versie...
Player player = GetPlayer(14);
//waarna we 1 van de overloads gebruik voor de logging
Logger.Write(player, categorie, prioriteit);
```

Codevoorbeeld 2.

```
try
{
  // code welke mogelijk een sql exception kan opleveren
}
catch(SqlException ex)
{
  bool rethrow = ExceptionPolicy.HandleException(ex, "Data Layer Policy");
  if (rethrow)
  {
    throw;
  }
}
```

Codevoorbeeld 3.

Logging

Het loggingblock is één van de blocks waarbij de vernieuwingen het best merkbaar zijn. Doordat het loggingblock nu gebaseerd is op de System.Diagnostics is het veel krachtiger en flexibeler geworden. Door de betere integratie met het .NET Framework kunnen we nu gebruik maken van classes als TraceListener en TraceSource in plaats van de Sinks die we in eerdere versies zagen. Een belangrijke vernieuwing qua flexibiliteit is dat we een LogEntry naar meer categorieën kunnen loggen. Voor deze categorieën kunnen we met behulp van filters ervoor zorgen dat onze logs niet te groot worden of dat we alleen de voor ons belangrijke informatie krijgen door bijvoorbeeld te filteren op de prioriteiten van LogEntries. In codevoorbeeld 2 zien we twee voorbeelden van het toepassen van logging.

Exception management

Met de komst van het .NET Framework is het een stuk eenvoudiger geworden om exception handling toe te passen in applicaties. Op een consistente manier exception management toepassen in de gehele applicatie is echter iets dat over het algemeen nog te weinig gebeurt. Het exception-handlingblock is de ideale manier om exception management op een simpele en consistente manier toe te passen in een applicatie. Door simpelweg een exceptie-policy te definiëren zijn we in staat om een exceptie te linken aan een actie. Zo kunnen we bijvoorbeeld een security-exceptie afvangen en als actie vervangen door een AccessDeniedExceptie. Of een SqlExceptie opvangen en deze als actie wrappen tot een exceptie van het type DataLaagExceptie. In codevoorbeeld 3 zien we hoe we een SQL-exceptie afhandelen met een policy en in afbeelding 2 is te zien hoe we deze configureren met de configuratieconsole.

ObjectBuilder

Zoals te zien is in afbeelding 1 is ObjectBuilder een van de Core-componenten van Enterprise Library. ObjectBuilder is een component dat buiten EntLib ook terug te vinden is in het Composite UI Application Block (CAB). Met behulp van ObjectBuilder is Patterns & Practices erin geslaagd om de afhankelijkheden die in afbeelding 1 te zien zijn daadwerkelijk los te koppelen.
ObjectBuilder is gebaseerd op het dependency injection pattern. Een uitleg van dit design pattern is te vinden op de site van Martin Fowler (zie de referenties). Hoewel je als gebruiker van EntLib niet direct contact zal hebben met de ObjectBuilder is het wel van essentieel belang dat je de principes hiervan kent als je van plan bent om EntLib uit te breiden. Dit artikel gaat niet diep genoeg in op ObjectBuilder om je deze informatie verschaffen. We willen echter een korte overview geven omdat dit de belangrijkste wijziging is tussen versie 1 en 2 van EntLib. Zoals de naam al zegt is ObjectBuilder verantwoordelijk voor het bouwen van objecten. Je maakt geen objecten meer aan via new, maar via factories, wat een ander design pattern is. De EnterpriseLibraryFactory gebruikt zelf weer ObjectBuilder. Nogmaals, bij het gebruik van EntLib zelf geldt dit niet, maar de core van EntLib zelf gebruikt dit mechanisme veelvuldig. Groot voordeel hiervan is dat ObjectBuilder op deze manier in staat is om afhankelijkheden te injecteren door gebruik te maken van constructor- of setter-injectie. Het gebruikt hiervoor attributen die de ObjectBuilder per object vertellen wat het object verwacht, zoals welke factory gebruikt moet worden of welke configuratie-instelling verwacht wordt. In Enterprise Library worden met behulp hiervan objecten gebuild die aan de hand van de configuratie goed geïnitialiseerd worden. Dit zonder dat de objecten zelf sterk afhankelijk zijn van deze configuratie. Het is in versie 2 van EntLib nu dan ook eenvoudiger om objecten zelf te construeren zonder gebruik te maken van de configuratie.

ObjectBuilder kent een pipeline of 'chain of responsibility', waarbij een aantal stappen wordt onderkend, namelijk:

- PreCreation
- Creation
- Initialization
- PostInitialization

Het is mogelijk om voor iedere fase in de pipeline een strategie (nog een design pattern) te schrijven die iets voor je doet tijdens één van de fases. Specifiek voor de diverse fases bevat ObjectBuilder een aantal strategieën dat interessante dingen kan doen zoals via de constructor een object maken, properties zetten, methodes aanroepen, reeds bestaande object-instanties teruggeven, enzovoort. Al met al een krachtig framework dat hopelijk ook als los block door Patterns & Practices uitgebracht zal worden met goede documentatie. ObjectBuilder wordt als onderdeel van EntLib uitgebracht onder een andere namespace dan bij CAB.

Migratie

Hoewel de API niet volledig compatibel is met eerdere versies van EntLib, zijn de wijzigingen klein waardoor het migreren eenvoudiger is. Bovendien heeft Microsoft migratie-guidance toegevoegd aan de EntLib-documentatie. Bij een migratie moeten we vooral letten op de belangrijkste zaken die in de nieuwe versie verdwenen of anders zijn. Zoals de wijziging van het security-applicationblock naar de ASP.NET-providers voor rollen, profielen en authenticatie en het configuratie-applicationblock dat vervangen is door de System.Configuration-namespace. Het formaat van de configuratiebestanden is helaas ook gewijzigd. In de praktijk zal dit betekenen dat bij gebruik van de configuratietool ook de hele configuratie opnieuw gemaakt moet worden. Vooral als je zelf nieuwe blocks hebt toegevoegd, zal dit het nodige werk meebrengen. Een voordeel is wel dat het maken van eigen configuratiesecties voor eigen blocks een stuk eenvoudiger is geworden!

Afbeelding 2. Een SQL-exceptie wordt geconfigureerd

Werkbesparing

Als je net begint met nieuwe projecten op basis van .NET Framework 2.0, dan kan je jezelf een hoop werk besparen door gebruik te maken van EntLib 2.0. Het configuration application block bestaat niet meer en configuratie is nu volledig op de System.Configuration-namespace gebaseerd. Het security application block is grotendeels vervangen door de Membership- en Profile-features die het framework nu standaard biedt. Helaas geen nieuwe blocks, maar wel een hoop verbeteringen om het ontwikkelen van nieuwe blocks te vergemakkelijken. Onderschat ook niet hoe eenvoudig je gebruik kunt maken van de instrumentation-features van EntLib. Ze zijn nu veel beter gedocumenteerd en je kunt hier veel profijt van hebben tijdens het analyseren van problemen in productieomgevingen van enterprise-applicaties die gebruikmaken van EntLib. Veel klanten vragen er niet direct om, maar verwachten het stiekem wel.

Mark Willems is Solution Developer bij Avanade (www.avanade.com), een samenwerkingsverband van Microsoft en Accenture. Voor vragen en opmerkingen is Mark te bereiken op markw@avanade.com of via zijn blog blog.markwillems.nl.

Dennis Mulder is principal consultant bij Avanade. Voor vragen en opmerkingen is Dennis te bereiken op dennism@avanade.com of via zijn blog www.dennismulder.net/blog.

Referenties

blogs.msdn.com/tomholl/
www.gotdotnet.com/codegallery/codegallery.aspxid=295a464a-6072-4e25-94e2-91be63527327
www.avanade.com
www.martinfowler.com/articles/injection.html
www.agileprogrammer.com/oneagilecoder/archive/2006/01/03/10564.aspx