Anko Duizer
Anko is werkzaam als trainer/coach bij Class-A. Daarvoor heeft hij 5 jaar gewerkt bij Microsoft als consultant. Onder zijn klantenkring bevinden zich voornamelijk Top100 bedrijven in Nederland. Sinds begin 2001 is hij bezig met .NET. Speciale interesse heeft Anko voor de architectuur en het ontwerp van een gedistribueerde applicatie.
anko.duizer@class-a.nl
www.class-a.nl

Asynchronous Invocation Application Block

Waarom en hoe het asynchronous invocation application block toe te passen?

In dit artikel behandelt Anko Duizer de volgende vraag: "wanneer en hoe pas ik het asynchronous invocation block toe?". Dit application block is recent gereleased door Microsoft. Het is gebaseerd op ervaringen en ontwerpreviews van succesvolle .NET-applicaties. Eerst wordt de 'waaromvraag' beantwoord. Vervolgens wordt het asynchronous invocation application block aan de binnenkant beschreven en wordt behandeld hoe eigen code te schrijven die gebruik maakt van het application block.

Wat is het probleem? De meeste applicatiecode maakt gebruik van synchrone aanroepen. Dit is een eenvoudig programmeermodel. Er zijn veel situaties denkbaar waarin het synchrone programmeermodel wenselijk dan wel noodzakelijk is. Er zijn echter ook situaties waarin asynchroon de voorkeur heeft. Programmeurs die de afgelopen jaren hebben gewerkt bij grotere organisaties hebben waarschijnlijk veelvuldig gebruik gemaakt van een asynchroon programmeermodel. Het grootste probleem van het synchrone model is namelijk dat je wacht op het antwoord, en zo lang je wacht kun je niets anders doen! Je staat 'geblockt'. Wanneer calls gemaakt gaan worden over de grenzen van applicaties en zelfs systemen heen is dit vaak niet het gewenste gedrag. Er is de laatste tijd ook de mogelijkheid bij gekomen om in de architectuur gebruik te maken van webservices. Webservices zijn van nature services die op een andere omgeving draaien. Wanneer je deze services aanspreekt vanuit een client is het wenselijk om dit asynchroon te doen. Dit geeft de gebruiker op z'n minst de illusie dat de applicatie een goede performance vertoont, omdat niet gewacht hoeft te worden op het antwoord voordat verder gegaan kan worden. Het zou bijvoorbeeld mogelijk zijn een webservice aan te spreken die een creditcardnummer checkt; dit kan asynchroon gebeuren. Het probleem van wachten speelt natuurlijk niet alleen bij clients, maar ook op de server of bij webservices zelf. Ook hier kan een asynchrone call helpen. Overigens heeft een asynchrone call meer voordelen dan alleen het voorkomen van 'blocking'. Het geeft de server de gelegenheid de call op te pakken wanneer de server daar de tijd voor heeft. Dit kan zeer voordelig zijn voor de schaalbaarheid van de server.

Wat is het Asynchronous Invocation Application Block?

Het asynchronous invocation application block heeft de doelstelling om de problemen van 'blocking' op te lossen. De documentatie spreekt expliciet van het scenario met een webbrowser en een webservice. In de praktijk is het application block echter veel breder inzetbaar; zowel de client als server kunnen praktisch alles zijn. Natuurlijk een webservice, maar ook een windows-client of windows-service. Het asynchronous invocation application block draagt zorg voor asynchrone communicatie tussen een (web)client en een of meer 'foreign service providers (FSP)'. Een FSP kan een .NET webservice, maar ook een Java webservice zijn.

Het application block biedt de infrastructuur een asynchrone call te maken vanuit een client en vervolgens te verwerken op een server. Om dit te kunnen doen is in ieder geval de volgende software vereist:

- Microsoft Windows 2000 of hoger
- Microsoft Internet Information Services (IIS) 5.0 of hoger
- Microsoft SQL Server Server 2000 (met SP3 of hoger)
- Microsoft .NET Framework (versie 1.0 of hoger)
- Microsoft Visual Studio.NET

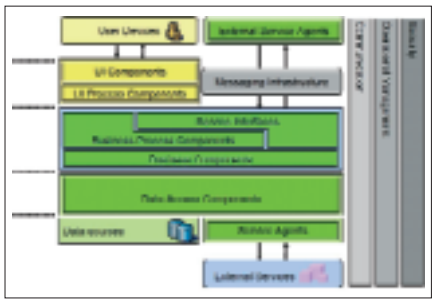
Component	Omschrijving
Request Subsysteem	Dit onderdeel zorgt ervoor dat het mogelijk wordt om vanuit de client op een eenvoudige manier een asynchrone call te starten, en vervolgens de resultaten op te halen.
Dispatcher Subsysteem	Dit is de kern van het application block. Het dispatcher onderdeel pakt de asynchrone aanvragen op en zorgt ervoor dat deze terecht komen bij de juiste service agent.
Monitor subsysteem	Dit onderdeel zorgt voor het opschonen van de database en het opruimen van service agents die blijven 'hangen'.

Tabel 1: Componenten in de architectuur van het asynchronous invocation application block

Het application block bestaat uit C#-code, VB.NET-code, een Quick Start en helpdocumentatie.

Hoe past het application block in de architectuur?

Het asynchronous invocation application block past perfect in de door Microsoft neergezette applicatiearchitectuur die is terug te vinden op de MSDN-site. In deze architectuur wordt expliciet gesproken over zogenaamde 'Service Agents'. Deze service agents worden in de architectuur gepositioneerd om te communiceren met externe webservices. Service agents staan in de architectuur op hetzelfde niveau als de data-sources. Ze leveren en persisteren data. Dit is weergegeven in afbeelding 1.



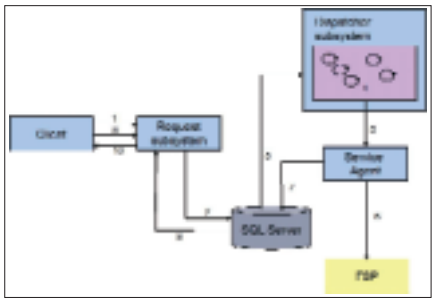
Afbeelding 1. .NET applicatiearchitectuur

Met het asynchronous invocation application block kun je het blok van de service agents perfect invullen. Het levert in feite de infrastructuur om eigen service agents te implementeren.

Wat is de interne architectuur?

Naast de applicatiearchitectuur is er ook een interne architectuur van het asynchronous invocation application block. De interne architectuur bestaat uit een drietal componenten, weergegeven in tabel 1.

Het asynchronous invocation block maakt intensief gebruik van een database, in het geval van Microsoft: SQL Server. In de database komen alle asynchrone aanvragen en de bijbehorende resultaten terecht. Daarnaast bevat de database metadata over wat er überhaupt aan asynchrone vragen kunnen worden gesteld; kortom, wat voor service agents zijn er beschikbaar?

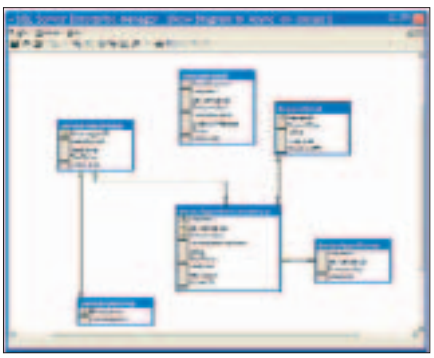


Afbeelding 2. Asynchronous invocation application block architectuur

In afbeelding 2 is de interne architectuur schematisch weergeven. Het mechanisme werkt als volgt:

1. Een client verzoekt het Request subsysteem om een specifieke asynchrone call. De client krijgt een request id terug.
2. Het Request subsysteem slaat het verzoek op in een Microsoft SQL Server database
3. Het Dispatcher subsysteem monitort de database, en haalt vervolgens het verzoek op
4. Het verzoek wordt toebedeeld aan een thread uit de pool
5. De thread start (op basis van metadata) de juiste server agent
6. De server agent plaatst het verzoek bij de desbetreffende FSP
7. Het resultaat wordt door de service agent bewaard in de SQL Server database
8. De client vraagt op basis van het request id om de resultaten
9. Het request subsysteem haalt de resultaten op uit de database
10. De resultaten worden terug gegeven aan de client

Het gebruikte databaseschema is eenvoudig; zie afbeelding 3. Het schema is een combinatie van metadata en 'productie'data. De metadata zijn terug te vinden in de tabellen: ServiceAgentMap en ServiceAgentMaster.



Afbeelding 3. Asynchronous invocation application block databaseschema

```
// Add the request to the Request Batch object
// Create request input for Deposit1
asyncReqBatch.AddRequest(
   "DepositInfo",
   "Deposit1",
   new object[]{
       CreateInputParam("AccountNo", "10001"),
       CreateInputParam("Amount", "100")});

// Deposit2
asyncReqBatch.AddRequest(
   "DepositInfo",
   "Deposit2",
   new object[]{
       CreateInputParam("AccountNo", "222000"),
       CreateInputParam("Amount", "323")});

// Submit the request
requestID = AsyncRequestProcessor.SubmitRequest(asyncReqBatch);
```

Voorbeeldcode 1. Request batch

```
public class CREDITCARDDETAILSServiceAgent :
Microsoft.ApplicationBlocks.AsynchronousInvocation.Dispatcher.
IServiceAgent
{

  /// <summary>
  /// Constructor to create the CREDIT CARD service agent.
  /// </summary>
  public CREDITCARDDDETAILSServiceAgent()
  {}

  /// <summary>
  /// The service agents provide implementation for the Execute
  /// method of the IServiceAgent interface.
  /// </summary>
  /// <param name="inputParamsArray">
  /// Array of Input arguments passed from the client code
  /// </param>
  /// <param name="CallBack">
  /// Callback used to return the results to the framework
  /// </param>
  public void Execute(
      Microsoft.ApplicationBlocks.AsynchronousInvocation.Dispatcher.
      WorkerThread.PersistCallBack CallBack,
      object[] inputParamsArray )
  {
      string results = "";
      Microsoft.ApplicationBlocks.AsynchronousInvocation.Common.
      ServiceAgentResult saResult;

                // Do the real work!!!
                // For example call a XML Web service

      // Provide the results
      results = "some string";

      // Create a results instance
      saResult = new Microsoft.ApplicationBlocks.
           AsynchronousInvocation.Common.ServiceAgentResult(results, "");


      // Callback to save the results in the database
      CallBack(saResult, true);
  }
}
```

Voorbeeldcode 2. Service Agent

Wat is de rol van het Request subsysteem?

Het Request subsysteem biedt een programmeermodel om asynchrone calls te kunnen starten vanuit een .NET-client, en vervolgens de resultaten te kunnen ophalen. In code wordt een 'batch' opgebouwd. Het is mogelijk om meer vragen in één batch te stellen. Wanneer de 'batch' is samengesteld, kan via de AsyncRequestProcessor een submit worden gedaan van het request. In codevoorbeeld 1 staat een voorbeeldbatch.

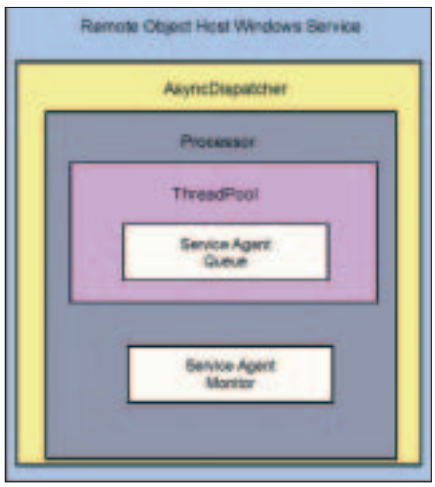
Het request subsysteem bestaat uit een drietal classes:

- AsyncRequestBatch
- AsyncRequestProcessor
- ResultsManager

Via de ResultsManager kunnen de resultaten worden opgehaald.

Wat is de rol van het Dispatcher subsysteem?

Het dispatcher subsysteem is de kern van het asynchronous invocation application block. In de basis zorgt dit deel van het systeem ervoor dat de gevraagde calls worden opgepakt en aan de juiste service agent worden gekoppeld. Op basis van de meegegeven informatie in de request batch wordt aan de desbetreffende service agent een 'schop' gegeven die vervolgens zijn werk gaat doen. Het dispatcher subsysteem zoekt de juiste service agent op basis van metadata uit de database. In codevoorbeeld 1 wordt in de code meegegeven dat het gaat op een DepositInfo, het dispatcher subsysteem vertaalt dit naar een service agent, die wordt geïmplementeerd als een assembly. Welke assembly's moeten worden geladen, is terug te vinden in de tabel ServiceAgentMaster. In afbeelding 4 staat een overzicht van de verschillende onderdelen die gezamenlijk het dispatcher subsysteem vormen. De implementatie van het dispatcher subsysteem draait als windows-service: AsyncROHService. Zodra de service is gestart, wordt een singleton instantie gemaakt van de class AsyncDispatcher. Vervolgens wordt een processor-instantie aangemaakt. Deze leest uit de metadata welke service agents beschikbaar zijn. Het dispatcher subsysteem is nu in leven.



Afbeelding 4. Dispatcher classes overview

Op een FIFO- basis (first in, first out) gaat de dispatcher calls uit de database lezen. De informatie in de verzoeken bepaalt welke service agent moet worden aange-

```
INSERT INTO dbo.ServiceAgentMaster(ServiceAgentId, AssemblyPath, ClassName, TimeToLive, RetryCount)
VALUES('{E5F9B082-DEF0-49a0-A3C6-EC41AF97D624}','C:\StarServiceAgent.dll',
'CLASSAServiceAgent.CREDITCARDDETAILSServiceAgent',90,2)
GO

INSERT INTO dbo.ServiceAgentMap(FriendlyName, ServiceAgentId) VALUES('AggregatedInfo',
'{E5F9B082-DEF0-49a0-A3C6-EC41AF97D624}')
GO
```

Voorbeeldcode 3. Registratiescript voor een service agent

sproken; dit wordt door de processor uitgezocht. Vervolgens wordt het verzoek in een in-memory queue van de desbetreffende service agent geplaatst. Zodra een thread beschikbaar is vanuit de thread pool wordt het verzoek door de service agent opgepakt uit de queue en uitgevoerd. De service agent praat op zijn beurt met een FSP, wat alles kan zijn. Dit is het punt dat bijvoorbeeld een creditcardcheck-webservice werkelijk wordt aangesproken. Zodra er resultaat is wordt dit door de service agent weggeschreven in de database. Het resultaat is door de client vervolgens op te halen op basis van een uniek request id. Het processorobject werkt met een poll-mechanisme. Eens in de zoveel tijd wordt de processor wakker en gaat kijken of er nieuwe verzoeken zijn. Is dit het geval dan start het proces opnieuw. De polling-interval is in te stellen via een configuratiefile.

Implementatie van een Service Agent

Wanneer een externe FSP moet worden aangesloten, dient een zogenaamde service agent te worden gemaakt. In de praktijk komt dit neer op het schrijven van een class die de interface IServiceAgent implementeert. Deze interface schrijft voor dat een methode Execute wordt geïmplementeerd. In deze methode wordt de werkelijke code geschreven om bijvoorbeeld een webservice aan te spreken.

In voorbeeldcode 2 is een mogelijke service agent in C# geprogrammeerd. Dit is een template voor een service agent. De werkelijke code om een FSP aan te spreken ontbreekt, maar dit is niet anders dan 'normale' code om bijvoorbeeld een webservice aan te spreken. In de code wordt gebruik gemaakt van een delegate voor de call-back om de resultaten terug te geven aan het asynchronous invocation block, om deze vervolgens in de database te plaatsen. Het is de taak van de client om de resultaten vervolgens op te halen. Hiervoor maakt de client gebruik van een uniek request id. Om de service agent te registreren bij het application block dienen twee tabellen in SQL Server te worden gevuld, namelijk ServiceAgentMaster en ServiceAgentMap. In voorbeeldcode 3 is een tweetal SQL-statements weergegeven om de service agent uit voorbeeld 2 te registreren.

De volgende drie configuratiefiles zijn aanwezig:

Configuratiefile	Omschrijving
MonitorService App.config	Deze file verzorgt de configuratie voor het monitor subsysteem. De volgende items kunnen worden geconfigureerd: · Connectie string · Time interval voor de garbage collector · Time interval voor de recovery service · Maximum aantal requests dat de garbage collector kan verwijderen in een slag.
ROHService App.config	Deze file verzorgt de configuratie voor het request subsysteem. Onder andere de volgende items kunnen worden geconfigureerd: · Connectie string · Aantal threads in de pool · Minimum aantal threads dat beschikbaar moet zijn in de thread pool · Time interval voor de dispatcher · Grootte van de queue
Client App.config	Deze file verzorgt de configuratie voor de client. De connectie string kan worden opgegeven.

Het application block implementeert een mechanisme om 'hangende' service agents op te ruimen. Dit is de taak van het monitor subsysteem. Wanneer de zogenaamde finish-time minder is dan de current time of het aantal retry counts minder is dan 1 wordt de desbetreffende call gemarkeerd als 'Failed'. Overigens is ook de gebruiker in staat een call af te breken. De ResultManager (onderdeel van het request subsysteem) biedt de mogelijkheid een call te af te breken. Vervolgens moet het monitor subsysteem de database opschonen.

Wat zijn de configuratiemogelijkheden?

Om het asynchronous invocation application block te configureren wordt gebruik gemaakt van een drietal .NET configuratiefiles. Het is mogelijk om zaken te configureren zoals databasetoegang, interval voor services zoals garbage collection, en de grootte van de thread pool.

Wat is de relatie met andere application blocks?

Het asynchronous invocation application block maakt standaard gebruik van twee andere blocks:

- Microsoft Data Application Block, dit applicatie block wordt toegepast om de Microsoft SQL Server op een eenvoudige en consistente manier te benaderen.
- Microsoft Exception Application Block. Dit application block wordt gebruikt om eventuele exceptions op een eenduidige manier te loggen in de event log.

Indien noodzakelijk is het asynchronous invocation application block te gebruiken met nog andere blocks. Denk bijvoorbeeld aan de Aggregation en Caching

application blocks. Wanneer data van verschillende FSP's moeten worden samengevoegd, biedt het aggregation application block de mogelijkheid om meer service agents aan te spreken; dit kan gebeuren via het asynchronous invocation application block. Het plaatje kan compleet worden gemaakt door het caching application block. Dit application block biedt de mogelijkheid om data te cachen. Wanneer de gewenste data niet aanwezig zijn, wordt de service agent aangesproken. Dat kan eventueel het aggregation block zijn. Met zijn drieën vormen deze application blocks een uitstekende infrastructuur voor bijvoorbeeld een website of webservice.

Uitstekende bouwsteen in iedere applicatiearchitectuur

Het asynchronous invocation application block zorgt voor de mogelijkheid om service agents asynchroon aan te roepen. Dit is een goed uitgangspunt voor een stabiele en schaalbare omgeving. Het asynchronous invocation application block is in diverse applicatiearchitecturen toe te passen. Overigens is het web een natuurlijke keuze om het asynchronous invocation application block toe te passen. De codevoorbeelden in dit artikel zijn geschreven in C#, Microsoft biedt het application block ook aan in VB.NET. Het is te gebruiken met alle .NET-talen. In combinatie met de andere application blocks is het asynchronous invocation block een uitstekende bouwsteen in iedere applicatiearchitectuur.

Referenties

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/paiblock.asp
http://msdn.microsoft.com/architecture

Microsoft patterns & practices

PROVEN PRACTICES FOR PREDICTABLE RESULTS

Patterns & practices bestaan uit specifieke aanbevelingen voor het ontwerpen, bouwen, implementeren en gebruiken van degelijke oplossingen voor complexe zakelijke en technische scenario's. Deze uitgebreide technische leidraad is op basis van praktijkervaringen samengesteld en voert dus veel verder dan een normale white paper.

Patronen en praktijkervaringen zijn interessant voor iedereen die te maken heeft met software. Ze helpen niet alleen bij het maken van een goede oplossing, maar kunnen ook veel tijd besparen. De meeste problemen die u in een project tegen komt zijn al een keer opgelost, dus waarom twee keer het wiel uitvinden?

Het gebruik van patronen en praktijkervaringen heeft onder meer de volgende voordelen:

- Bewezen - gebaseerd op praktijkervaring
- Autoriteit - beste architectuuradvies beschikbaar
- Accuraat - technisch gevalideerd en getest
- Direct toepasbaar - bevat duidelijke stappen
- Actueel - oplossingen voor hedendaagse problemen
- Relevant - geen virtueel scenario, maar gebaseerd op een bewezen oplossing.

Binnen Microsoft en bij partners van Microsoft wordt veel software ontwikkeld. Om snellere, goedkopere en betere software te kunnen ontwikken, wordt van elk project bijgehouden wat er goed en fout is gegaan en wordt dit zoveel mogelijk gecommuniceerd. Als de ervaringen interessant genoeg zijn, worden deze verzameld en samengevat in een patterns & practices.

De patterns & practices zijn onderverdeeld in drie categorieën en zijn interessant voor softwareachitecten, ontwikkelaars en beheerders.

- Naslagwerken over architectuur
- Naslagwerken over infrastructuur
- Naslagwerken over building blocks & IT-services

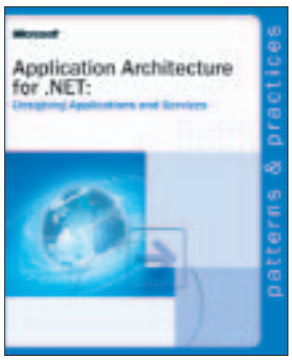
Een groot aantal van de ervaringen en patronen - die direct in de praktijk toepasbaar zijn - hebben wij op de cd patterns & practices gezet. Deze kan gratis worden besteld bij Microsoft Nederland. Inmiddels zijn er meer dan 50 patterns & practices beschreven.

Patterns & practices

Voor een compleet overzicht van de patterns & practices en de laatste informatie bezoekt u de patterns & practices site: www.microsoft.com/resources/practices/
Patterns & practices zijn ook in boekvorm beschikbaar en te bestellen bij:
http://www.knowledgecentre.nl/series/index.asp?keyword=patterns
De patterns & practices CD-Rom is te bestellen op:
www.microsoft.com/netherlands/msdn/patternspractices.asp
De patterns & practices worden regelmatig behandeld in Microsoft webcast: www.microsoft.com/resources/practices

Application Architecture for .NET: Designing Applications and Services

Deze gids biedt softwarearchitecten en ontwikkelaars die met het Microsoft® .NET Framework gedistribueerde oplossingen willen bouwen ondersteuning op architectuur- en op ontwerpniveau . Er wordt verondersteld dat u ervaring hebt met het ontwikkelen van .NET-componenten en de basisbeginselen voor het ontwerp van gelaagde gedistribueerde applicaties onder de knie hebt. Bent u verantwoordelijk voor het opzetten van de architectuur en het ontwerp van applicaties of services? Moet u anderen adviseren over de juiste technologieën en producten voor applicaties of services? Neemt u ontwerpbeslissingen die aansluiten op de (non-)functionele behoeften in uw bedrijf? Of is het uw taak om de juiste communicatiesystemen voor applicaties of services te kiezen? Dan is deze gids voor u een must.



Titel: Application Architecture for .NET: Designing Applications and Services
ISBN: 0735618372