Sander Gerz
is ontwikkelaar bij Sogeti Nederland BV, bestuurslid van de stichting dotNed en initiatiefnemer van DevTips.NET, een Nederlandstalige website voor ontwikkelaars over Microsoft .NET
sander@devtips.net

Exceptions en Events

Nieuwe mogelijkheden voor fout- en eventhandling

Het Microsoft .NET Framework biedt de Visual Basic programmeur nieuwe mogelijkheden voor fout- en eventhandling. In dit artikel beschrijft de auteur hoe deze nieuwe mogelijkheden met behulp van exceptions en delegates met elkaar kunnen samenwerken. Event- en exceptionhandling lijken op het eerste gezicht niet veel met elkaar te maken te hebben.

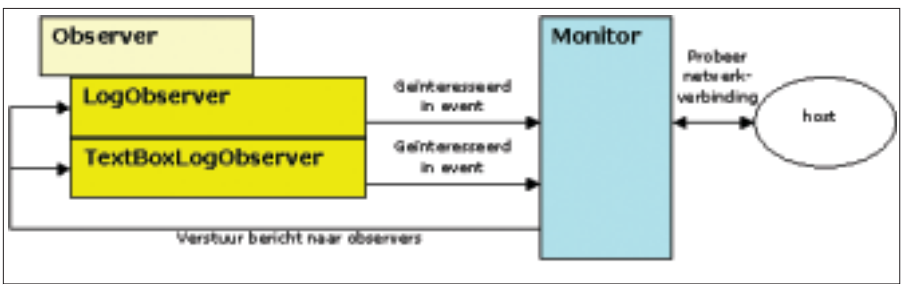
In dit artikel gaan we echter kijken naar een combinatie van events en exceptions. Dat doen we aan de hand van een voorbeeldapplicatie waarvan de complete broncode ook beschikbaar is (zie nuttige internetadressen; red.). De applicatie controleert met een in te stellen interval op de beschikbaarheid van een computer. Zodra de applicatie via het netwerk geen verbinding meer kan maken met deze computer op deze poort wordt dit gemeld aan één of meer objecten.

In dit artikel gebruiken we Visual Basic.NET voor de voorbeelden. Deze broncode van de voorbeeldapplicatie is ook geschreven in Visual Basic .NET en is te downloaden op www.devtips.net.

Structured versus Unstructured Exception Handling

In Visual Basic 6.0 maakte je gebruik van unstructured exception handling wanneer je fouten wilde afvangen in je code. Je zette hiervoor aan het begin van een blok het On Error statement en elke fout die optrad in dit blok werd onderschept. In Visual Basic .NET is het nog steeds mogelijk gebruik te maken van unstructured exception handling, maar dat wordt sterk afgeraden. Er is immers een zeer goed alternatief, namelijk: structured exception handling. Dit mechanisme zit in de CLR en is dus in VB.NET te gebruiken. Een exception is iedere fout of onverwachte situatie die optreedt tijdens het uitvoeren van een programma. Exceptions kunnen onder andere optreden door een fout in je programmacode of externe code die je aanroept, door een gebrek aan systeembronnen en fouten die de common language runtime ontdekt, bijvoorbeeld wanneer code niet geverifieerd kan worden.

Ook exceptions die in een andere .NET-taal zijn opgetreden kunnen door VB.NET worden afgehandeld. Het is daarom van belang dat iedere .NET-programmeur in staat is met exceptions om te gaan. Om verwerking van exceptions mogelijk te maken moeten de instructies gevat zijn in een try-catch blok of een try-catch-finally blok (zie Voorbeeldcode 1. Een try-catch-finally blok).



Afbeelding 1. Monitor voorbeeldapplicatie

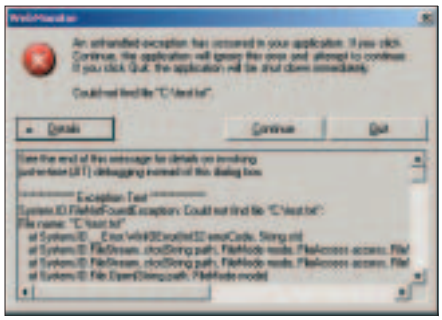
Try

In het try-blok wordt de code geplaatst die je wilt controleren op het optreden van fouten. Als je geen try-catch blok opneemt rondom deze code en er treedt een fout op, dan wordt de exception doorgegeven aan de code die de methode heeft aangeroepen. Dit proces gaat net zo lang door totdat de .NET Runtime een procedure mét exceptionhandling tegenkomt. Als er geen enkele procedure met exceptionhandling wordt gevonden, dan handelt de .NET Runtime de exception zelf af (zie afbeelding 2).

Het tonen van exception-informatie door de .NET Runtime is onwenselijk. Zo bevat het meldingsscherm een Continue-button die gevaarlijke consequenties kan hebben. Ook wil je niet dat een gebruiker dergelijke detailinformatie te zien krijgt.

Catch

Zodra een exception optreedt wordt een instantie van de System.Exception class of een afgeleide hiervan gemaakt. Het Exception-object is in het Catch-statement als parameter mee te geven. Je kunt zo



Afbeelding 2. Exceptionhandling door de .NET Runtime

onder andere de message-eigenschap van het Exception-object bekijken om meer te weten te komen over de oorzaak van de exception (zie voorbeeldcode 2). Wanneer een exception via een catch-statement is afgehandeld verdwijnt dit Exception-object. Het kan voorkomen dat de methode die de foute code heeft aangeroepen ook op de hoogte wil worden gebracht van het optreden van een exception. Dit is mogelijk door het throw-keyword.

Finally

Op het moment dat een applicatie bepaalde resources aanspreekt is het van belang dat die resources na gebruik ook weer worden losgelaten. Bij het optreden van exceptions – maar ook zonder – moeten bestanden en connecties gesloten worden. Het Finally-statement geeft je de mogelijkheid dit te realiseren. Code die in het Finally-blok staat wordt altijd uitgevoerd, ongeacht het wel of niet optreden van een exception.

Hiërarchie

Zoals gezegd zijn alle exceptions afgeleid van de System.Exception class. De hiërarchie in deze classes werkt door in de afhandeling van exceptions. Je moet eerst controleren op specifieke exceptions en daarna de meer generieke; een exception wordt immers maar één keer afgehandeld. Wanneer je het try-catch blok zou opzetten zoals in Voorbeeldcode 3, dan zou sommige code nooit kunnen worden uitgevoerd. De compiler waarschuwt overigens niet voor deze constructie. Het is dus van belang de hiërarchie te kennen van de Exceptions-classes. Je kunt deze in Visual Studio.NET vinden via Ctrl+Alt+E.

```
Try
   ‘voer code uit waarin mogelijk een exception kan optreden
   ‘bijvoorbeeld: het uitvoeren van een Sql query
Catch
   ‘vang de exception op
   ‘bijvoorbeeld: meldt dat de query niet correct is uitgevoerd.
Finally
   ‘deze code wordt altijd uitgevoerd; exception of niet
   ‘bijvoorbeeld: sluit de connectie naar de database
End Try
```

Voorbeeldcode 1. Een try-catch-finally blok

```
Catch ex As System.IO.IOException
‘The exception that is thrown when an I/O error occurs.
Dim msg As String
msg = “Er is een schrijffout opgetreden waardoor niet naar het “
msg += “logbestand kan worden bijgewerkt.” & vbCrLf
msg += “De geconstateerde fout is: “ & vbCrLf
msg += ex.Message & vbCrLf
msg += “De bron is: “ & vbCrLf
msg += ex.Source
Console.WriteLine(msg)
```

Voorbeeldcode 2. Het Catch-statement uitgebreid

```
Try
   ‘schrijf naar een logbestand
Catch ex As System.Exception
   ‘er is een exception opgetreden; alleen deze code wordt uitgevoerd
Catch ex As System.IO.IOException
   ‘hier komt de CLR nooit omdat System.Exception generieker is
Catch ex As System.IO.PathTooLongException
   ‘hier komt de CLR nooit omdat System.Exception generieker is
End Try
```

Voorbeeldcode 3. Verkeerde volgorde van exceptionhandling

Wanneer gebruik je exceptions

Gezien de kracht van exceptionhandling in .NET zou je tot de conclusie kunnen komen om overal in de broncode try-catch blokken te plaatsen. Zorg er wel voor dat je niet met exceptions werkt voor de normale werking van een applicatie. Exceptions kunnen namelijk de performance van een applicatie behoorlijk omlaag halen. Als het mogelijk is via andere manieren controles uit te voeren is dat zeker aan te bevelen. Hierbij kan worden gedacht aan controles op nullwaarden, aanwezigheid van bestanden en de inhoud van variabelen alvorens er berekeningen mee uit te voeren.

Samenvattend zijn de voordelen van exceptionhandling als volgt:

- Je kunt gebruik maken van het Exception-object dat niet alleen informatie bevat over de fout die op dat moment is opgetreden, maar ook de fouten toont die hieraan ten grondslag liggen.
- Het is mogelijk te erven van de Exception-classs waardoor je eigen exceptions kunt maken met dezelfde functionaliteit als de baseclass of zelfs meer.
- Omdat elke class in het .NET Framework exceptions kan opwerpen zodra er een runtime-fout optreedt zullen ontwikkelaars het steeds gewoner vinden om exceptionhandling toe te voegen aan hun code.
- Je kunt try-catch blokken nesten in andere try-catch of try-catch-finally blokken. Zodoende kun je exceptionhandling zo globaal of specifiek maken als je zelf wilt.
- Exceptions werken over talen heen

Exceptions en events

Je weet nu dat exceptions kunnen optreden en hoe je deze kunt afhandelen. De

informatie die een Exception-object bevat kan interessant zijn voor verschillende groepen. De benodigde informatie verschilt echter per doelgroep. Een gebruiker wil bijvoorbeeld wel weten dat een fout is opgetreden, maar heeft niets aan cryptische berichten met een call-stack, regelnummers en foutcondities. Algemene foutmeldingen in de trant van 'Er is een fout geconstateerd, waarschuw de helpdesk' zijn overigens voor de gebruiker ook vrij irritant. Bovendien kunnen exceptions voor ontwikkelaars en systeembeheerders van belang zijn, maar voor deze groepen zijn vaak andere en meer details relevant. Om deze redenen zou je een applicatie kunnen uitrusten met een Exception Managementsysteem dat op zichzelf geïmplementeerd wordt, zonder impliciete aannames over de wijze waarop exceptions worden gelogd of gerapporteerd. Deze koppeling tussen een exception en diens behandeling kan worden losgemaakt met behulp van events. Zo kan een applicatie gemonitored worden en kunnen gebruiker, systeembeheerders en ontwikkelaars op een andere wijze op de hoogte worden gebracht van eventuele problemen. Om dit te realiseren moet je weten hoe events werken.

Events

Elke programmeur die een Windows-applicatie ontwikkelt, maakt gebruik van events. Voorbeelden van events zijn een klik op een button of menu-item, het bewegen van de muis over een label en tekst invoeren in een textbox. Het event is een bericht dat wordt gestuurd door een object naar andere objecten om te melden dat zich iets heeft voorgedaan. Het object dat het bericht verstuurt wordt de event sender genoemd, terwijl het object dat het bericht ontvangt de event receiver is. In de event receiver moet een methode zitten die automatisch wordt uitgevoerd zodra het een bericht ontvangt.
In het .NET Framework is het de programmeur erg makkelijk gemaakt om event-driven te programmeren. Zo eenvoudig zelfs, dat je niet hoeft te weten hoe het principe eigenlijk werkt. Wanneer je in Visual Studio .NET dubbelklikt op een control die je op een form hebt geplaatst, wordt namelijk automatisch code toegevoegd voor

```
'declaratie van een delegate
Public Delegate Sub MonitorEventHandler(ByVal sender As Object, ByVal e As
MonitorEventArgs)
```

Voorbeeldcode 4. Declaratie van een delegate

```
| |___[CLS] MonitorEventHandler
| | |   .class public auto ansi sealed
| | |   extends [mscorlib]System.MulticastDelegate
| | |___[MET] method .ctor : void(object,native int)
| | |___[MET] method BeginInvoke : class [mscorlib]System.IAsyncRe-
              sult(object,class WebMonitor.WebMonitor.MonitorEvent-
              Args,class [mscorlib]System.AsyncCallback,object)
| | |___[MET] method EndInvoke : void(class [mscorlib]System.IAsyncResult)
| | |___[MET] method Invoke : void(object,class WebMonitor.WebMoni-
              tor.MonitorEventArgs)
```

Voorbeeldcode 5. De delegate in Intermediate Language

```
'1. declaratie van de delegate
Public Delegate Sub MonitorEventHandler(ByVal sender As Object, _
       ByVal e As MonitorEventArgs)

'2. De eventargs class
Public Class MonitorEventArgs
        Inherits EventArgs

        Private _monitorException As Exception

        Public Sub New(ByVal monitorFailure As Exception)
            Me._monitorException = monitorFailure
        End Sub

        Public Property MonitorException() As Exception
            Get
                Return _monitorException
            End Get

            Set(ByVal Value As Exception)
                _monitorException = Value
            End Set
        End Property

End Class

'3. het event als instantie van een delegate
Public Event MonitorFailure As MonitorEventHandler

'4. de methode waarin het event wordt opgeroepen
Protected Overridable Sub OnFailure(ByVal e As MonitorEventArgs)
     RaiseEvent MonitorFailure(Me, e)
End Sub

'5. het event kan op verschillende plaatsen optreden
Dim failureEvent As New MonitorEventArgs(ex)
OnFailure(failureEvent)
```

Voorbeeldcode 6. Events stap voor stap

```
Public Class Observer
   'basic implementation
    Public Overridable Sub MonitorFailed(ByVal sender As Object, _
           ByVal e As MonitorEventArgs)
    End Sub
End Class
```

Voorbeeldcode 7. De Observer class

een eventhandler en de registratie van de eventhandler in de eventsource. Events vormen echter een centraal onderdeel van de CLR en het is dan ook belangrijk om te weten wat achter een event schuilgaat. Laten we daarom eens kijken hoe je zelf een event kunt maken.

Observer pattern

Voor ons event werpen we even een blik op het zogenoemde Observer design pattern uit de wereld van objectgeoriënteerd programmeren (Erich Gamma, e.a.). Het observer pattern kent twee belangrijke objecten: de observer en het subject. Een subject kan meer observers hebben. Deze observers luisteren of het subject een notificatie verstuurt van een wijziging in het subject. Dit Observer pattern kan gebruikt worden in de volgende situaties:

1. een wijziging in het ene object zorgt voor een wijziging van andere object, maar je weet van tevoren niet hoeveel objecten gewijzigd moeten worden.
2. een object moet in staat zijn berichten naar andere objecten te sturen zonder dat het object weet wie die objecten zijn. Anders gezegd, je wilt dat deze objecten niet vastgekoppeld zijn.
3. een systeem kent twee aspecten waarbij de één afhankelijk is van de ander. Wanneer je deze aspecten in twee aparte objecten onderbrengt kun je deze gemakkelijker en onafhankelijk van elkaar hergebruiken.

Tijdens de communicatie in het geval van events weet de event sender niet welk object of methode het bericht zal ontvangen. Er is een intermediair nodig tussen de bron en de ontvanger. Deze intermediair is in het .NET Framework van het type Delegate.

Delegates

De delegate is een typesafe, op class gebaseerde, referentie naar een methode. Een delegate heeft de volgende kenmerken:

- een delegate is een class
- een delegate is typesafe
- je kunt verscheidene delegates combineren in één delegate
- je kunt delegates gebruiken voor statische en niet-statische methoden
- je kunt delegates definiëren binnen en buiten een class
- delegates kun je gebruiken voor asynchrone applicatiecode
- delegates worden vaak gebruikt voor events.

Met name vanwege het laatste kenmerk gaan we in dit artikel delegates gebruiken. In Voorbeelcode 4 zie je hoe een delegate gedeclareerd kan worden.

Alhoewel het lijkt alsof hier een subroutine wordt gedeclareerd, zorgt deze regel in werkelijkheid voor de declaratie van een class. Je kunt dat bijvoorbeeld zien als je met ildasm.exe naar de gecompileerde applicatie kijken (zie Voorbeeldcode 5). Met ildasm.exe zie je dat de delegate eigenlijk vertaald wordt naar een publieke class MonitorEventHandler() die erft van System.MulticastDelegate.

Het event stap voor stap

De basis voor onze eventhandler is gelegd. In onze voorbeeldapplicatie is het de bedoeling dat een exception optreedt, zodra geen netwerkverbinding gemaakt kan worden met een opgegeven computer. Het Exception-object wordt via een event doorgegeven aan die objecten die in dit event geïnteresseerd zijn. In Voorbeeldcode 6 zie je hoe de Visual Basic code er uit ziet voor de stappen die je moet nemen.

1. We beginnen met de declaratie van de delegate. De delegate heeft de naam MonitorEventHandler gekregen en kent twee argumenten: sender en e. Het tweede argument is van het type MonitorEventArgs. We definiëren MonitorEventArgs in stap 2.
2. MonitorEventArgs is een class waarin de data zit die de event sender doorgeeft aan de event receiver. De data die we in het event doorsturen bestaat slechts uit een Exception-object. Hier komt de combinatie van events en exceptions al om de hoek kijken.
3. In onze applicatie hebben we een Monitor-class met één event: MonitorFailure. In de declaratie van dit event zien we dat het event een instantie creëert van onze delegate.
4. In onze Monitor-class declareren we vervolgens een methode waarvan de naam begint met 'On'. In deze methode wordt het event opgeroepen. Het event kent twee argumenten. Dat is logisch, want het event is een instantie van de delegate die bij het instantiëren om deze twee argumenten vraagt. Het eerste argument is de sender (Me) en het tweede argument is een MonitorEventArgs-object.
5. Tot slot is het zaak dat we ergens in onze Monitor-class het event ook werkelijk aanroepen. De locatie hiervan is natuurlijk afhankelijk van de vraag op welk moment we het nodig vinden dat van iets melding moet worden gemaakt. In ons geval ligt het voor de hand om het event aan te roepen zodra er geen verbinding meer gemaakt kan worden met de opgegeven computer. Stap 4 en 5 lijken makkelijk te combineren tot één. Dat is ook zo, maar de ontwerpregels stellen hier een scheiding voor.

Luisteren naar events

Nu is eigenlijk nog maar de helft geïmplementeerd: een object met een event dat kan worden opgeroepen. Het heeft echter weinig zin om een bericht te sturen als niemand luistert. Je hebt een object nodig die interesse heeft in het event. In het voorbeeld is dit de class Observer.

Deze Observer kent één methode MonitorFailed die dezelfde argumenten kent als het event dat wordt opgeroepen. Deze methode is overridable omdat er in het voorbeeld subclasses gebruikt worden met een implementatie van de MonitorFailed-methode.

Er wordt een instantie gecreëerd van deze class (Dim obs As New Observer()). Zodra we beschikken over het object kan de methode (MonitorFailed) aan het event (MonitorFailure) worden gekoppeld via: AddHandler mon.MonitorFailure, AddressOf obs.MonitorFailed

Op deze manier is het mogelijk om in

runtime te kiezen of een object wel of niet luistert naar een bepaald event. Met AddHandler wordt de interesse kenbaar gemaakt en de methode RemoveHandler zorgt voor de loskoppeling van het event en de methode.

In de voorbeeldapplicatie zijn er twee subclasses die erven van de Observer-class. Deze subclasses schrijven bij het optreden van events naar een tekstbestand en naar een tekstbox. Je kunt natuurlijk ook meer of andere objecten maken die het optreden van het event op een andere wijze behandelen.

Exceptions: nieuw en krachtig principe

Exceptions zijn voor de Visual Basic-ontwikkelaar één van de nieuwe en zeer krachtige principes van het .NET Framework. Zorg er wel voor dat je niet met exceptions werkt voor de normale werking van een applicatie. Exceptions kunnen namelijk de performance van een applicatie behoorlijk omlaag halen. Ook de gedetailleerde informatie in exceptions zijn niet voor iedereen even interessant. Een applicatie moet wel in staat zijn om exception-informatie op te slaan. Als de gegevens over het optreden van een exception niet netjes worden bewaard, dan is het lastig of zelfs onmogelijk om na te gaan om welke reden exceptions optreden. Dit geldt met name voor applicaties zonder user interface.

Door gebruik te maken van events voor het versturen van de exception-informatie ben je flexibel in het bouwen en toepassen van objecten die exceptions afhandelen. In het .NET Framework kun je gebruik maken van delegates om dergelijke events te implementeren.

Noten

1. De voorbeeldapplicatie is te downloaden op http://www.devtips.net. Relevante links zijn ook te vinden op deze website.
3. Design Patterns, Erich Gamma e.a., Addison-Wesley Pub Co; ; 1st edition (15 januari 1995), ISBN 0201633612
4. Een uitgebreide implementatie van het mechanisme dat hier wordt beschreven, is te vinden in het Exception Management Application Block, een standaard applicatieblok dat Microsoft ter beschikking stelt om direct in eigen applicaties toe te passen.