Heindirk de Laat is softwarearchitect bij Atos Origin

.NET Framework 2.0 compatibiliteit

HOE VOORKOM JE INCOMPATIBILITEIT BIJ WINDOWS- EN WEBAPPLICATIES?

Microsoft heeft compatibiliteit hoog in het vaandel staan. Na security komt compatibiliteit samen met betrouwbaarheid op de tweede plaats. Eind maart gaf Microsoft zijn klanten de kans de compatibiliteit van bestaande .NET-applicaties te testen met het op handen zijnde .NET Framework 2.0. In vier dagen werden onder begeleiding van Microsoft-medewerkers compatibiliteitsproblemen opgespoord en besproken. In dit artikel leg ik het belang van compatibiliteit uit en hoe problemen met compatibiliteit in het .NET Framework kunnen worden voorkomen.

De .NET-applicaties zijn gecompileerd met een bepaalde versie van het .NET Framework. Dat wil niet zeggen dat deze applicaties per definitie met die versie van het .NET Framework worden uitgevoerd. Met de komst van het .NET Framework 2.0 is het van belang te weten hoe de versie van de .NET runtime wordt bepaald en wat je als programmeur kunt doen om je code zoveel mogelijk geschikt te maken voor volgende versies van het .NET Framework.

Wat is compatibiliteit?

Er wordt onderscheid gemaakt tussen twee vormen van compatibiliteit.

- Backward compatibility is het vermogen van code, die is gebouwd met een bepaalde versie van de runtime, om te draaien op een *nieuwere* versie van die runtime.
- Forward compatibility is het vermogen van code, die is gebouwd met een bepaalde versie van de runtime, om te draaien op een *oudere* versie van die runtime. Dit wordt niet ondersteund door versie 2.0 van het .NET Framework. De rest van dit artikel gaat over backward compatibility.

Wat is incompatibiliteit?

Er wordt onderscheid gemaakt tussen drie vormen van incompatibiliteit.

- Er is sprake van *binary incompatibility* wanneer de publieke interface van een assembly is gewijzigd zodat applicaties niet langer gebruik kunnen maken van dezelfde interfaces.
- Er is *source code incompatibility* wanneer een applicatie gebruik kan blijven maken van een service terwijl deze is gewijzigd, maar de code kan niet opnieuw worden gecompileerd tegen de nieuwere versie van de service. Bij het laden van een oude solution converteert Visual Studio 2005 de code naar het nieuwe .NET Framework. Doordat versie 2.0 op sommige punten flink afwijkt, kan een handmatige conversie toch noodzakelijk zijn.
- Tot slot is er *behavioral incompatibility*. Iedere wijziging die resulteert in een gedragswijziging van een bestaande applicatie leidt tot behavioral incompatibility.

Welke versie?

Welke versie van het .NET Framework wordt gekozen, hangt af van een aantal dingen. Er is een groot verschil tussen de werking van Windows- en webapplicaties.

Windows, console, Windows service-applicaties

Een applicatie wordt standaard uitgevoerd door de runtime waarmee het is gecompileerd. Als managed code wordt gecompileerd (.exe of .dll) dan wordt de versie van het .NET Framework geregistreerd in de assembly. Als die runtime niet aanwezig is, maar een latere versie wel, dan wordt de latere versie van de runtime gebruikt. Managed components die worden aangeroepen vanuit COM-componenten via COM Interop, en managed components die worden aangeroepen vanuit een common host (dllhost, svchost of bijvoorbeeld Internet Explorer), gebruiken de recentste versie van de runtime die aanwezig is op de machine. Managed components die worden aangeroepen vanuit een native applicatie kunnen een bepaalde versie van de runtime aangeven via hosting API's. Dit is vergelijkbaar met het specificeren via configuratiebestanden; zie het volgende punt. In het App.config-bestand kan de ontwikkelaar aangeven welke versies van de runtime kunnen worden gebruikt. De volgorde in het configuratiebestand bepaalt de volgorde waarin naar de runtimes wordt gezocht. Dit werkt ook als de managed code wordt aangeroepen vanuit een native exe of vanuit COM.

Zo zijn op een machine bijvoorbeeld de runtimes aanwezig van versie 1.1 en 2.0. In codevoorbeeld 1 wordt runtime 1.1 gebruikt en in codevoorbeeld 2 runtime 2.0. Dit geldt overigens alleen als de applicatie is gebouwd met versie 1.0 of 1.1 - omdat 2.0 van de runtime niet forward compatibel is met versie 1.1.

Als geen enkele opgegeven runtime aanwezig is, faalt de executie. Het supportedRuntime-element vervangt het requiredRuntime-element dat beschikbaar was bij het uitkomen van versie 1.1. Het requiredRuntime-element is nog wel nodig om applicaties die met versie 1.1 zijn gecompileerd te laten draaien onder versie 1.0.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v1.1.4322" />
    <supportedRuntime version="v2.0.50215" />
  </startup>
</configuration>
```

Codevoorbeeld 1.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v2.0.50215" />
    <supportedRuntime version="v1.1.4322" />
  </startup>
</configuration>
```

Codevoorbeeld 2.

Vermijdt incompatibiliteit. Zorg dat je applicatie gebruik maakt van het .NET Framework waarmee het is gecompileerd

Het specificeren van de runtimes in het configuratiebestand mag alleen worden gedaan als de startende applicatie ook eigenaar is van de managed code die wordt aangeroepen. Het wordt beschouwd als slechte gewoonte om dit te doen voor externe applicaties omdat daarmee mogelijk de applicatie faalt. Soms is het echter de enige manier om de juiste versie van de runtime aan te wijzen.
Als managed code wordt aangeroepen vanuit een native exe of COM-component dan zijn er twee mogelijkheden. Alle managed code draait onder de meest recente versie van de runtime; dit is het standaard gedrag. Of alle managed code draait onder de versie die is gespecificeerd in het configuratiebestand.

ASP.NET-, webservices-applicaties
Bij ASP.NET-applicaties wordt de versie van de runtime alleen bepaald door de mapping (script map) die in IIS is vastgelegd tussen extensies en het .NET Framework. Per webapplicatie en per virtual directory kan deze mapping worden vastgelegd. Als niets wordt vastgelegd, wordt de mapping van de parent directory gebruikt.

De ontwikkelaar kan de mappings op een aantal manieren wijzigen:
- Run `aspnet_regiis -s w3svc/1/root` (aspnet_regiis staat in de directory van de runtime waarvoor de webapplicatie moet worden geconfigureerd; voor iedere versie is er een beschikbaar). Een website kun je configureren door deze toe te voegen aan de opdracht (bijvoorbeeld `aspnet_regiis -s w3svc/1/root/WebApplication1`).
- Gebruik de ASP.NET-tab in de properties van de webapplicatie in IIS (dit is nieuw met de komst van versie 2.0 van de runtime en dit is hetzelfde als het gebruik van `aspnet_regiis`).
- Installeer versie 1.1 van de runtime op een machine die alleen versie 1.0 van de runtime bevat (dit is hetzelfde als het uitvoeren van `aspnet_regiis` op het root level). Het installeren van versie 2.0 van de runtime verandert de mappings niet.

Om te controleren hoe de mappings zijn geconfigureerd kun je de volgende opdracht uitvoeren (het maakt niet uit welke versie van aspnet_regiis wordt gebruikt):

```
aspnet_regiis -lk
```

Opmerking: het is in Visual Studio .NET 2003 voor ASP.NET-applicaties mogelijk om de supported runtimes in te stellen. Deze optie wijzigt weliswaar het webconfiguratiebestand, maar wordt bij het uitvoeren genegeerd.

Wat breekt potentieel de compatibiliteit?

De volgende opsomming is niet volledig, maar geeft een goed idee waarmee je rekening moet houden bij het schrijven van compatibele code. 1) Het opslaan van informatie in de directory van een specifieke versie van de runtime (zoals machine.config of security.config). 2) Het parsen van .NET Framework-strings. Als je bijvoorbeeld de tekst van een exception doorgeeft of het versienummer van de runtime, dan bestaat de kans dat deze code met een volgende versie van de runtime niet meer werkt. 3) Het doen van aannames over het gedrag van het .NET Framework. Je mag geen aannames doen over bijvoorbeeld de volgorde van reflection, enumerations, unordered collections en de volgorde waarin finalizers worden uitgevoerd. Verder is het onverstandig uit te gaan van waardes van hashcodes of de geheugenlay-out van managed processen. 4) Het communiceren tussen processen waarbij sommige managed zijn en sommige worden geladen via COM. Als deze applicaties met elkaar communiceren via serialisatie dan is de kans groot dat dit fout gaat. 5) Applicaties die objecten serialiseren met de ene versie van de runtime en deserialiseren met een andere versie. Dit is vergelijkbaar met de het vorige punt. In feite bestond dit probleem al tussen de versies 1.0 en 1.1. Een handvol classes uit het .NET Framework 1.1 werd anders ge(de)serialiseerd dan in 1.0. In versie 2.0 is het aantal wijzigingen veel groter en daarmee eveneens de kans dat problemen ontstaan. Een deel van de serialisatieproblemen kun je oplossen door het gebruik van Version Tolerant Serialization (VTS). VTS maakt het mogelijk types te serialiseren tussen het .NET Framework 1.1 en 2.0. Hier is wel een patch voor 1.1 nodig. Voorlopig werkt deze oplossing alleen bij binary formatters. Door VTS is het mogelijk types uit het .NET Framework - die gewijzigd zijn - te (de)serialiseren tussen versie 1.1 en 2.0. Het kan ook worden gebruikt om problemen op te vangen indien eigen types worden gewijzigd - tussen twee versies van de applicatie. In het verleden bracht een wijziging altijd een onherstelbare fout mee, tenzij de serialisatie was geïmplementeerd met ISerializable. Het gebruik van VTS maakt de implementatie van ISerializable vrijwel overbodig. 6) Het gebruik van DateTime-waarden in combinatie met een XmlSerializer. In versie 1.1 van het Framework werd bij serialisatie altijd de tijdzone bij een DateTime geplaatst. In versie 2.0 hangt dit af van het gebruik van DateTime. DateTime.Now voegt de tijdzone toe, het gebruik van new DateTime voegt geen tijdzone toe en het gebruik van DateTime.ToUniversalTime voegt de letter Z toe. 7) Applicaties die communiceren via .NET Remoting waarbij verschillende versies van de runtime worden gebruikt. 8) Applicaties die obfuscation gebruiken. 9) Applicatie-setups die testen op de aanwezigheid van een specifieke versie van het .NET Framework. 10) Applicaties die testen op een specifieke versie van het .NET Framework bij het uitvoeren. 11) Code die zich anders gedraagt in verschillende versies van het .NET Framework zoals bugfixes; behavioral incompatibility.

Het testen van een applicatie

Als een applicatie moet draaien onder het .NET Framework versie 2.0 moet je de keuze maken tussen het uitvoeren van een conversie of testen op compatibiliteit. Als je een computer hebt waarop de runtime-versies 1.1 en 2.0 staan, kun je een applicatie testen door een specifieke runtime op te geven. In code kun je testen welke versie van de runtime wordt gebruikt met behulp van System.Environment.Version.

Je kunt je applicatie testen door de volgende scenario's te gebruiken.
- Test de applicatie op een machine waar alleen versie 1.1 is geïnstalleerd; en zorg dat dit goed werkt.
- Test de applicatie op een machine waar versie 1.1 en 2.0 is geïnstalleerd en draai deze applicatie met 1.1. Hiermee test je eventuele bijwerkingen van het installeren van versie 2.0.
- Test de applicatie op een machine waar versie 1.1 en 2.0 is geïnstalleerd en draai deze met 2.0.
- Tot slot: test de applicatie op een machine waar alleen versie 2.0 is geïnstalleerd.

Je kunt de applicatie configureren om een speciale versie te gebruiken. Bij *Windows-applicaties* gebeurt dat als volgt. Configureer de applicatie door het toevoegen van het supportedRuntime-element in het app.config-bestand. Of gebruik 'the big red switch' in de registry. Door deze switch te gebruiken zullen alle applicaties op een machine de meest recente versie van de runtime gebruiken; standaard gedrag. Deze setting is alleen beschikbaar voor testdoel-

einden en is onderdeel van de bèta 1 en bèta 2 releases: [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework]"OnlyUseLatestCLR"=dword:00000001
Bij *webapplicaties* gaat het zo: Wijzig de mappings met `aspnet_regiis.`

Conclusies

Als je incompatibiliteit wilt vermijden, zorg er dan voor dat je applicatie gebruik maakt van het .NET Framework waarmee het ook is gecompileerd. Voor webapplicaties is dit eenvoudig. Plaats de juiste versie van de runtime op de webserver en configureer de applicatie om deze te gebruiken. Het is goed mogelijk om ASP.NET-applicaties met een nieuwere versie te draaien, maar er is altijd een bepaald risico. Voor Windows-applicaties is het soms niet mogelijk of wenselijk om de juiste versie te bepalen. Dat wil zeggen dat de applicatie getest moet worden op compatibiliteit.

Heindirk de Laat werkt voor Atos Origin als softwarearchitect. Hij nam van 28 tot en met 31 maart deel aan het Compatibility DevLab bij Microsoft in Redmond. Zijn e-mailadres is heindirk.delaat@atosorigin.com

Nuttige internetadressen

Side-by-Side Execution of the .NET Framework: http://msdn.microsoft.com/library/en-us/dnnetdep/html/sidexsidenet.asp
Configuring an ASP.NET Application for an ASP.NET Version: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconconfiguringaspnetapplicationforaspnetversion.asp
Side-By-Side Execution for COM Interop: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconSide-By-SideExecutionForCOMInterop.asp
Version Tolerant Serializing: http://msdn.microsoft.com/msdnmag/issues/04/10/AdvancedSerialization/default.aspx