ASP.NET

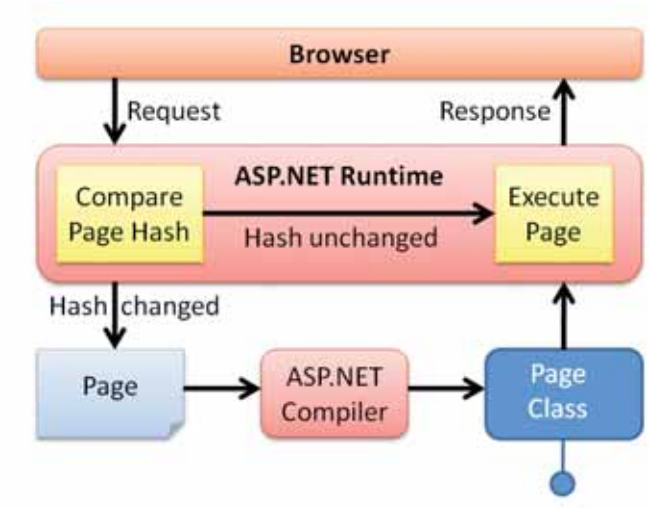
VirtualPathProvider to the limit

Michiel van Otegem

De VirtualPathProvider is een niet zo bekend onderdeel van ASP.NET. Wie er wél bekend mee is, ziet het vaak als een nuttige feature voor SharePoint. Met de VirtualPathProvider kun je echter meer dan je op het eerste gezicht zou denken. Bijvoorbeeld een handig mechanisme maken om ASP.NET-pagina's en User Controls te delen tussen verschillende projecten.

De VirtualPathProvider grijpt op een fundamenteel niveau in op de manier waarop ASP.NET een pagina-aanvraag afhandelt en valt daardoor voor de meeste ontwikkelaars buiten hun blikveld. Als ontwikkelaars er al bekend mee zijn, passen ze de VirtualPathProvider toch niet vaak toe. Dit heeft twee redenen: men ziet het nut er niet van in en op het eerste gezicht lijkt de VirtualPathProvider redelijk complex, met het risico moeilijk traceerbare bugs sneller te introduceren. De complexiteit blijkt echter een stuk minder als je er wat beter in duikt. Hiervoor is het wel zeer belangrijk dat je weet hoe ASP.NET aanvragen voor bestanden (pagina's, controls, etcetera) precies afhandelt en welke rol de VirtualPathProvider hierin speelt. In grote lijnen ben je er vast wel bekend mee, maar de details zijn in dit geval des te belangrijker.

Wanneer we een ASP.NET-pagina voor het eerst opvragen, moet het .aspx-bestand nog gecompileerd worden naar Intermediate Language (IL), voordat de Common Language Runtime de pagina daadwerkelijk kan uitvoeren. Als de code-behind niet al in een assembly zit, moet ASP.NET de code-behind op dat moment ook compileren. Het compileren van de pagina levert een assembly op die ASP.NET plaatst in %SystemRoot%\Microsoft.NET\Framework\<versie>\Temporary ASP.NET Files. Daarin staat voor elke gecompileerde pagina een assembly (en dat kunnen er dus heel wat worden). Voor iedere volgende aanvraag voor de betreffende pagina, voert ASP.NET de pagina uit vanuit die assembly, ook bij een applicatie-herstart. Wel is het uiteraard zo dat, zolang er geen applicatie-herstart plaatsvindt, de assembly in het geheugen zit als deze eenmaal door de JIT-compiler tot machinetaal is omgezet. Wanneer de oorspronkelijke pagina verandert, verwijdert ASP.NET de bestaande assembly en begint het hele proces opnieuw. Of een pagina is aangepast, ziet ASP.NET aan de bestandsnaam van de assembly die het gemaakt heeft. Die is namelijk gebaseerd op een voor iedere pagina unieke hashcode, die niet verandert zolang de pagina niet wijzigt. Het hele proces ziet er daardoor uit als in afbeelding 1.



AFBEELDING 1. AANVRAAGAFHANDELING IN ASP.NET

Standaard VirtualPathProvider

Het berekenen van de hashcode, waarmee ASP.NET bepaalt of de pagina opnieuw gecompileerd moet worden, is de taak van de GetFileHash()-methode van de VirtualPathProvider. De standaard VirtualPathProvider berekent de hashcode op basis van de creatiedatum, de laatste wijzigingsdatum en de grootte van het bestand, plus die van alle bestanden waarvan de pagina afhankelijk is, zoals het codebestand voor de code-behind (als deze ook dynamisch gecompileerd wordt). Standaard gebruikt ASP.NET de MapPathBasedVirtualPathProvider. Het is een goed idee deze eens te bekijken met Reflector. Je ontdekt dan nog een aantal classes die een rol spelen, namelijk MapPathBasedVirtualFile en MapPathBased VirtualPath, die respectievelijk erven van VirtualFile en VirtualPath. Deze classes vormen de basis voor het virtuele file system dat de VirtualPathProvider aanbiedt. Dit is weergegeven in afbeelding 2.

Zoals je ziet in afbeelding 2 biedt de VirtualPathProvider de bestanden aan op basis van het virtuele pad binnen de website. Met de MapPathBased VirtualPathProvider komen deze van het fysieke file-systeem. 'Virtual' in VirtualPathProvider heeft echter een dubbele betekenis: het zorgt voor een virtueel file-systeem voor de

Je kunt met VirtualPathProvider niet voor een bepaalde map een eigen provider aanwijzen. Het is zinloos meer providers van hetzelfde type te registreren.

ASP.NET Runtime. Hoe de virtuele bestanden en mappen opgeslagen zijn, doet voor de ASP.NET Runtime niet ter zake, zolang ze maar aangeboden worden via classes die erven van VirtualFile en VirtualPath. De signatuur van deze classes en de base class waarvan deze erven zie je in codevoorbeeld 1. De belangrijkste methode is eigenlijk VirtualFile.Open(), die een Stream retourneert. Die Stream representeert voor ASP.NET het te compileren bestand. Bij het implementeren van een VirtualPathProvider moet je die Stream dus opbouwen vanuit de bron waarin je de bestanden opslaat.

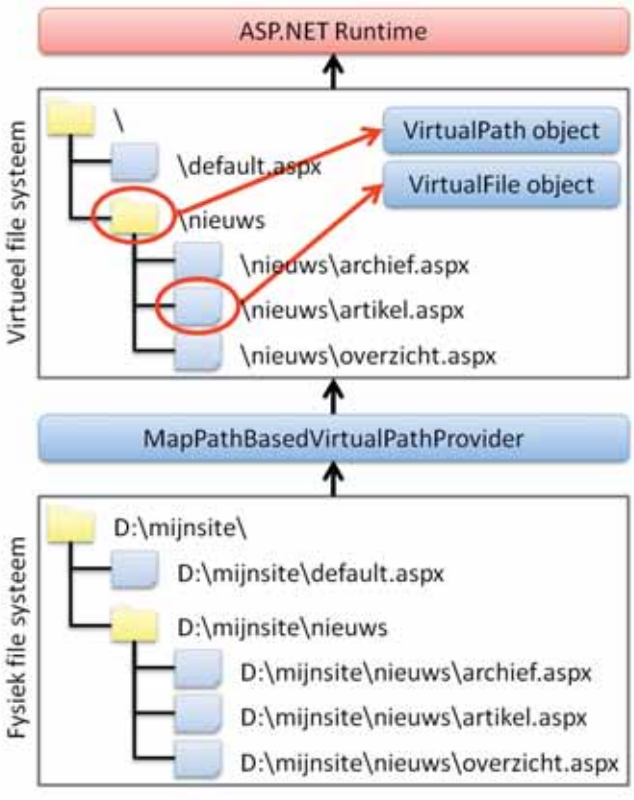
```
public abstract class VirtualFileBase : MarshalByRefObject
{
   public override object InitializeLifetimeService();

   public abstract bool IsDirectory { get; }
   public virtual string Name { get; }
   public string VirtualPath { get; }
}

public abstract class VirtualFile : VirtualFileBase
{
   protected VirtualFile(string virtualPath);
   public abstract Stream Open();

   public override bool IsDirectory { get; }
}

public abstract class VirtualDirectory : VirtualFileBase
{
   protected VirtualDirectory(string virtualPath);

   public abstract IEnumerable Children { get; }
   public abstract IEnumerable Directories { get; }
   public abstract IEnumerable Files { get; }
   public override bool IsDirectory { get; }
}
```

CODEVOORBEELD 1. VIRTUALFILE EN VIRTUALPATH CLASSES

ASP.NET gebruikt dus standaard de MapPathBasedVirtualPathProvider en deze blijft van toepassing, tenzij je een andere provider registreert. Daarbij vervang je deze echter niet, maar registreer je een VirtualPathProvider met een hogere prioriteit. Je registreert een VirtualPathProvider in de Application_Start()-methode in Global.asax. Dit doe je als volgt:
HostingEnvironment.RegisterVirtualPathProvider(new MyVirtualPathProvider);
Omdat we een VirtualPathProvider registreren in Global.asax ligt het voor de hand dat je deze niet kunt aanbieden vanuit je eigen provider. Dan ontstaat immers een kip-ei probleem. Daarnaast geldt als vuistregel dat je via de VirtualPathProvider alleen bestanden kunt aanbieden die zichtbare output produceren, zoals pagina's, User Controls, Master Pages en de daaraan verwante code. Tenzij je een BuildProvider levert voor typen bestanden die je wilt kunnen ondersteunen. Web.config is dus bijvoorbeeld niet geschikt om op te slaan in een andere bron (naast het feit dat dit zeer waarschijnlijk niet handig is). Je mag verschillende providers registreren, waarbij de laatst geregistreerde provider de hoogste prioriteit heeft. Elke VirtualPathProvider werkt vanaf de root van de website. Je kunt dus niet voor een bepaalde map een eigen provider aanwijzen. Hierdoor is het zinloos meer providers van hetzelfde type te registreren, tenzij je meer bronnen hebt. Een beter alternatief is in dat geval echter om de provider die bronnen te laten combineren.
Als een gevraagd bestand niet beschikbaar is via de VirtualPathProvider met de hoogste prioriteit, vraag je via de eigenschap VirtualPathProvider.Previous aan de provider met lagere prioriteit of deze het bestand wel kent. Hierdoor haal je bestanden in één map uit verschillende bronnen. Let op dat je zelf providers met een lagere prioriteit moet aanroepen om bestanden via die providers te kunnen aanbieden. Als jouw provider alleen kijkt naar het eigen "file systeem", kun je hier niet van profiteren. Verder moet je in je achterhoofd houden dat als een bestand met dezelfde naam in meer bronnen voorkomt, het bestand van de provider met de hoogste prioriteit wint. Een pagina op het fysieke file-systeem komt dus altijd pas als laatste, tenzij je daar zelf wat aan doet.

Bekende providers

SharePoint heeft een eigen VirtualPathProvider die pagina's leest uit een database. Hoe je zelf een dergelijke VirtualPathProvider maakt, lees je op http://support.microsoft.com/kb/910441. Dit gebruik van de VirtualPathProvider is met name interessant voor



AFBEELDING 2. KOPPELING TUSSEN HET FYSIEKE EN VIRTUELE FILE-SYSTEEM MET DE MAPPATHBASEDVIRTUALPROVIDER

Content Management-systemen, omdat de content dan in de page class gecompileerd wordt en daardoor slechts eenmalig uit de database gehaald hoeft te worden. Dit levert al snel een aanzienlijke performancewinst op. Daarnaast biedt deze aanpak ook de mogelijkheid ASP.NET controls of code in het CMS toe te staan. Hoewel je daar vanuit veiligheidsoverwegingen wél voorzichtig mee moet zijn. Een database is echter lang niet de enige manier waarop je met een VirtualPathProvider een virtueel file-systeem kunt aanbieden. Op http://msdn.microsoft.com/en-us/library/aa479502.aspx vind je een voorbeeld van gebruik van een Zip-bestand om bestanden in op te slaan. Zo zijn er nog allerlei voorbeelden te bedenken, de ene nuttiger dan de ander.

Omdat een assembly na compilatie geen mappen kent is de praktijk iets weerbarstiger

Embedded Resources als opslag

Een wat minder voor de hand liggende manier om bestanden op te slaan is als Embedded Resource in een assembly. Dit heeft twee belangrijke voordelen voor het uitrollen van pagina's en User Controls. Ten eerste kun je pagina's en User Controls op deze manier eenvoudig delen tussen diverse applicaties. Ten tweede biedt het de mogelijkheid goed versiebeheer toe te passen op deze elementen. Je kunt aparte assemblies maken met daarin de pagina's en User Controls die je wilt uitrollen en deze linken in het project waarin dit moet gebeuren. Wanneer je een nieuwe versie maakt, hoef je alleen de nieuwe assembly uit te rollen. Dit is vooral handig voor pagina's en controls die onderdeel uitmaken van standaardfunctionaliteit, zoals beheerpagina's voor een Content Management-systeem. Dat soort pagina's wil je niet voor ieder project in je website kopiëren, want dan wordt versiebeheer een drama. Door deze pagina's op te nemen in een assembly behoort dit soort problematiek tot het verleden.

Je kunt bestanden opnemen in een assembly als Embedded Resource door het betreffende bestand op te nemen in een Visual Studio-project en dan Embedded Resource als Build Action in te stellen. Vervolgens is het verzorgen van de VirtualFile.Open()-methode kinderspel, omdat de Assembly.GetManifestResourceStream()-methode een Stream retourneert. En dat is precies wat je met de Open()-methode moet doen. Het enige wat je hoeft te doen is het virtuele pad koppelen aan het bestand in de assembly. Wanneer je bestanden als Embedded Resource in een assembly zet, krijgen ze een unieke naam, namelijk de namespace waarin ze zitten, gevolgd door een punt en de naam van het bestand. Dus als we een library NetMag.Demo zouden maken met daarin een map Admin en daarin een bestand Default.aspx, luidt de volledige naam NetMag.Demo.Admin.Default.aspx. Houdt er rekening mee dat dit hoofdlettergevoelig is. Een simpele versie van de VirtualFile.Open()-methode zou er dan uitzien als in codevoorbeeld 2. Het voorbeeld gaat er echter vanuit dat de resources in dezelfde assembly staan als de VirtualFile-code en dat is niet handig als je deze VirtualPathProvider wilt hergebruiken.

Zelfs als de klant toegang heeft tot de server, loop je niet het risico dat iemand wijzigingen heeft aangebracht in de pagina's en User Controls

```
public Stream OPen(String virtualPath)
   String fullName;
   Assembly assembly = Assembly.GetExecutingAssembly();

   fullName = String.Format("{0}.{1}", assembly.GetName().Name,
   virtualPath.Replace("/", ".");
   return assembly.GetManifestResourceStream(fullName);
}
```

CODEVOORBEELD 2. VERSIMPELDE VIRTUALFILE.OPEN()-METHODE BIJ LEZEN UIT EEN ASSEMBLY

Omdat een assembly na compilatie geen mappen kent is de praktijk iets weerbarstiger. Codevoorbeeld 2 gaat dan ook alleen werken voor simpele gevallen. De code in de download bouwt eerst een soort sitemap op in een Dictionary, zoals te zien in codevoorbeeld 3. In dit codevoorbeeld worden met Assembly.GetManifestResourceNames() alle embedded resources opgevraagd in een assembly. Op basis van de configuratie wordt vervolgens het volledige virtuele- en fysieke pad bepaald en opgeslagen in de Dictionary. Daardoor kun je op basis van het pad de bijbehorende resource vinden. Hierbij houdt de EmbeddedResourceVirtualPathProvider rekening met verschillende assemblies waaruit pagina's zijn te halen. Deze assemblies stel je in web.config in met de configuratie uit afbeelding 3. Ieder element verwijst naar een assembly. Die assembly koppel je met VirtualRoot aan een specifieke virtuele map waarvoor de pagina's in de assembly van toepassing zijn. Ook kun je aanhaken op een specifieke plek in de mappenstructuur in de assembly door een ResourcePath te specificeren. Geef je bijvoorbeeld VirtualRoot de waarde \Admin en ResourcePath de waarde \UserAdmin, dan wordt \UserAdmin\Default.aspx in een assembly in de website \Admin\Default.aspx. Je bent daardoor heel flexibel om in de configuratie te bepalen op welke virtuele plek in de website bestanden uit een assembly staan. Dezelfde assembly mag dus meer keren voorkomen in de configuratie, aangezien je verschillende mappen op verschillende plaatsen in de site kunt zetten.

```
private void FillResourceDictionary()
{
   // Loop door alle geconfigureerde assemblies.
   foreach (ResourceStoreInfo resourceStoreInfo in m_Resource-
   StoreInfoList)
   {
      // Vraag alle embedded resources op en bepaal de virtuele
      paden.
      String[] resources = resourceStoreInfo.Assembly.GetManifest-
      ResourceNames();
      for (int i = 0; i < resources.Length; i++)
      {
         String virtualPath = TranslateResourceNameToVirtualPath
         (resourceStoreInfo, resources[i]);
         ResourceInfo resourceInfo = new ResourceInfo()
         {
            PhysicalPath = resources[i],
            ResourceStoreInfo = resourceStoreInfo,
            VirtualPath = virtualPath
         };

         if (!m_ResourceDictionary.ContainsKey(virtualPath.ToLower()))
         {
            m_ResourceDictionary.Add(virtualPath.ToLower(),
            resourceInfo);
         }
      }
   }
}
```

CODEVOORBEELD 3. EEN DICTIONARY VULLEN OM HET VIRTUELE FILE-SYSTEEM TE REPRESENTEREN

```
<EmbeddedResourceVirtualPathProvider>
   <ResourceItems>
      <ResourceItem Assembly="NetMag.Demo.dll" VirtualPath="\
      Admin" />
   </ResourceItems>
</EmbeddedResourceVirtualPathProvider>
```

AFBEELDING 3. CONFIGURATIE VAN DE EMBEDDEDRESOURCEVIRTUALPATHPROVIDER

Om zo flexibel met de virtuele bestanden om te gaan, bevat de de Dictionary die de sitemap representeert objecten van het type ResourceInfo. Daarin staan de assembly waarin het bestand te vinden is en het fysieke pad binnen de assembly. Hierdoor kunnen we opvragen zonder allerlei toeren uit te halen met het virtuele pad. Die translatie tussen het virtuele pad en het fysieke pad, die je ziet in codevoorbeeld 4, gebeurt per bestand eenmalig bij het vullen van de Dictionary.

```
private String TranslateResourceNameToVirtualPath(ResourceStore-
Info resourceStoreInfo, String resourceName)
{
   // Bepaal wat er van het volledige fysieke pad af moet.
   // De assemblynaam gaat er standaard vanaf, want dat is de
   container.
   String pathToRemove = resourceStoreInfo.Assembly.GetName().
   Name;
   if (String.IsNullOrEmpty(resourceStoreInfo.ResourcePath) ==
   false)
   {
      pathToRemove += NamespaceSeparator + resourceStoreInfo.
      ResourcePath.Replace(PathSeparator, NamespaceSeparator);
   }

   String virtualPath = resourceName;
   if (virtualPath.IndexOf(pathToRemove) > -1)
   {
      // Maak van het volledige fysieke pad een virtueel pad.
      virtualPath = resourceName.Remove(0, pathToRemove.Length);
      virtualPath = virtualPath.Replace(NamespaceSeparator,
      PathSeparator);
      virtualPath = virtualPath.Substring(0, virtualPath.
      LastIndexOf(PathSeparator)) + NamespaceSeparator + virtual-
      Path.Substring(virtualPath.LastIndexOf(PathSeparator) + 1);
   }

   // Koppel de resource aan een specifieke map in de website.
   if (String.IsNullOrEmpty(resourceStoreInfo.VirtualRoot) == false)
   {
      virtualPath = resourceStoreInfo.VirtualRoot + virtualPath;
   }

   return virtualPath.ToLower();
}
```

CODEVOORBEELD 4. TRANSLATIE TUSSEN HET VIRTUELE PAD EN HET FYSIEKE RESOURCE-PAD

De code-behind kun je in de assembly opnemen en gewoon laten compileren

Deze Dictionary wordt constant gebruikt om te bepalen of het bestand of de gevraagde map ook daadwerkelijk bestaat. Om dit optimaal te kunnen doen is de key van de Dictionary het virtuele pad in lower case, zodat het niet uitmaakt hoe de URL in de browser is ingetikt. De Dictionary zorgt ervoor dat je niet bij iedere aanvraag de gevraagde Embedded Resource op hoeft te zoeken. Dat is door de hoofdlettergevoeligheid lastig. Bovendien zitten in de assembly geen mappen en moet je de mappenstructuur dus op een of andere manier extraheren.

Echt praktisch?

Je vraagt je nu wellicht af of de EmbeddedResourceVirtualPathProvider in de praktijk echt werkt. Het antwoord is een volmondig ja. Doordat ASP.NET alleen bestanden via de provider nodig heeft op het moment dat deze gecompileerd moeten worden, is de performance-impact minimaal. Deze zit immers alleen in de controle of een bestand bestaat en zo ja of deze ongewijzigd is. Zolang de assembly niet wijzigt, is dat uiteraard het geval. Bestanden in een Embedded Resource krijgen, is ook niet echt heel moeilijk. Maak de pagina's en User Controls aan in een 'gewone' website en sleep ze in je assembly. Daarna hoef je de .aspx- en .ascx-bestanden alleen nog te markeren als Embedded Resource. De code-behind kun je in de assembly opnemen en gewoon laten compileren. Deze aanpak zorgt bij het uitrollen van nieuwe versies voor een minimum aan kopzorgen over pagina's die misschien in het doelproject gewijzigd zijn, want dat kan niet. Zelfs als de klant toegang heeft tot de server, loop je niet het risico dat iemand wijzigingen heeft aangebracht in de pagina's en User Controls. Alle visuele wijzigingen kunnen alleen via CSS en Themes. En voor andere talen werk je bij voorkeur met resource files.

Michiel van Otegem, is Chief Software Architect bij BataviaLabs.