What's new in ASP.NET 4.0?

Nieuwe impuls voor al zeer krachtig framework

Patrick Smits

Met de komst van ASP.NET 4.0, in combinatie met Visual Studio 2010, zal Microsoft het al zeer krachtige ASP.NET framework een nieuwe impuls geven. De verbeteringen binnen ASP.NET zijn op verschillende vlakken terug te vinden. Zowel binnen WebForms, ASP.NET Ajax als binnen MVC (Model View Controller) zijn talrijke verbeteringen doorgevoerd. Het ontwikkelen en deployen van ASP.NET-applicaties is nog nooit zo eenvoudig geweest.

Met de volgende versie van ASP.NET heeft Microsoft ervoor gekozen om de verschillende frameworks, die als aparte download beschikbaar waren en al onderdeel waren van het standaard ASP.NET Framework, te integreren tot één ASP.NET framework. Zo zijn nu zowel MVC, ASP.NET Ajax als het Dynamic Data Framework een geïntegreerd onderdeel van het ASP.NET 4.0 framework (afbeelding 1).

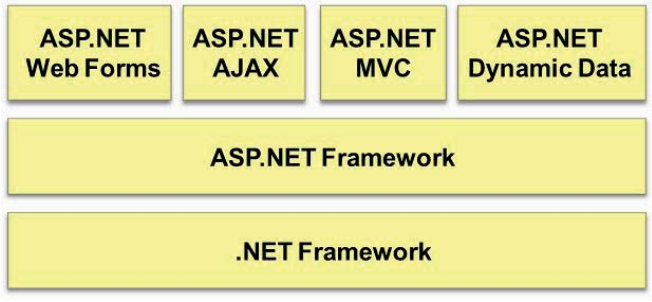
Web.config veranderingen

Wanneer je een nieuwe webapplicatie start, zie je al direct één van de belangrijkste wijzigingen in het ASP.NET 4.0 framework als het gaat om het beheren van de applicatie. De standaard web.config (van een lege webapplicatie) bestaat nu nog maar uit negen regels (zie codevoorbeeld 1).

```
<?xml version="1.0"?>
<configuration>
    <system.web>
        <compilation debug="false" target-
Framework="4.0" />
    </system.web>
    <system.webServer>
      <modules runAllManagedModulesForAllR
equests="true"/>
    </system.webServer>
</configuration>
```

CODEVOORBEELD 1.

In oudere versies van het framework werden de http-handlers voor bijvoorbeeld de Ajax-extension in de web.config geregistreerd. Dit had tot gevolg dat een standaard web.config 126 regels lang was. Wanneer je vervolgens ook nog gebruik wilde maken van bijvoorbeeld MVC kwamen daar nog handlers bij. Aangezien men bij de meeste webapplicaties gebruik wil maken van minimaal Ajax en soms ook nog van MVC, heeft Microsoft ervoor gekozen om deze 'standaard' http-handlers niet meer in de web.config te registreren, maar in de machine.config van het .NET Framework. Gevolg is dat de grootte van de web.config per webapplicatie enorm zal worden gereduceerd, omdat deze instelling uit de machine.config zal worden gehaald. De web.config zal dus een relatief schone configuratiefile zijn wat het beheer van deze configuratiefiles uiteraard ten goede zal komen.



FIGUUR 1.

Webforms

Vrijwel iedere webapplicatie, geschreven in ASP.NET, maakt gebruik van webforms. In ASP.NET 4.0 zijn er een aantal belangrijke wijzigingen doorgevoerd in dit framework. De belangrijkste wijziging is het zelf kunnen definiëren van de ClientIDs voor de webcontrols. In de vorige versies van ASP.NET werden de clientIDs gegenereerd door het .NET framework. Dit was echter een read-only property. Wanneer je vervolgens in de browser middels Javascript een object op wilde zoeken in het DOM op basis van de naam van het object, had je twee mogelijkheden. De eerste mogelijkheid is het gebruik van het attribuut ClientID van het webcontrol (zie codevoorbeeld 2).

```
<html xmlns="http://www.w3.org/1999/xht-
ml">
<head>
    <script language="javascript">
        window.onload = function() {

document.getElementById("btnAlert").on-
click = function() {

alert(document.getElementById("<% = txtUs-
erName.ClientID %>").value);
            }
        };
    </script>
</head>
<body>
```

```
    <form id="form1" runat="server">
        <asp:TextBox ID="txtUserName"
runat="server"> </asp:TextBox>
        <input id="btnAlert" type="button"
value="Click" />
    </form>
</body>
</html>
```

CODEVOORBEELD 2.

Dit is wel van negatieve invloed op de leesbaarheid en de beheersbaarheid van de code. Je kunt bijvoorbeeld dit soort constructies niet (eenvoudig) in .JS-files gebruiken, aangezien deze files niet worden verwerkt door de ASP.NET runtime. De tweede optie is het hard coderen van de naam van het control in Javascript. Nadeel hiervan is dat dit snel tot fouten kan leiden wanneer de mark-up van de pagina zal veranderen. In codevoorbeeld 2 zal de ClientID 'txtUsername' zijn. Wanneer we vervolgens de mark-up zullen wijzigen, door bijvoorbeeld gebruik te maken van een masterpage, kan de clientID wijzigen naar 'ctl00_ContentPlaceHolder1_txtUserName' Gevolg is dat de Javascript-code niet meer zal werken. Het zou dus mooi zijn wanneer het mogelijk is om de naam van een webcontrol op de client wat beter te kunnen voorspellen en dat deze niet zal wijzigen wanneer de mark-up wijzigt. Dit is precies wat de nieuwe property 'ClientIDmode' doet.

```
<%@ Page Language="C#"
ClientIDMode="Static"
AutoEventWireup="true"
MasterPageFile="~/MasterPage.master"
CodeFile="Default.aspx.cs" Inherits="_De-
fault" %>

<asp:Content ID="mainContent"
runat="server"
    ContentPlaceHolderID="mainContentPlace
Holder">
  <!—ContentPlaceHolderId is gedefinieerd
in de masterpage 
    <asp:TextBox ID="txtUserName"
runat="server"> </asp:TextBox>
 </asp:Content>
```

CODEVOORBEELD 3.

Deze property kan de waarde AutoID, Static, Predictable of Inherit bevatten. Daarbij zal de waarde van het ClientID dus variëren van een kopie van de naam van het ID van het control (Static) tot een volledig voorspelbare ID in het geval het een datacontrol betreft (perdictable).
Zoals uit voorgaande voorbeelden blijkt, is dit een enorm krachtige toevoeging voor het ASP.NET framework. Het maakt het selecteren en bewerken van serverside-gegenereerde HTML-elementen vanuit clientside Javascript (bijvoorbeeld middels JQuery) stukken eenvoudiger.

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title></title>
    <script language="C#" runat="server">

protected void Page_Load(object sender,
EventArgs e)
        {
            Dictionary<string, string> da-
talist =

new Dictionary<string, string>()
            {   {"6010","Klant 1"},
                {"9988","Klant 2"},
                {"6765","Klant 3"}
            };
            lsvKlanten.DataSource = data-
list;
            lsvKlanten.DataBind();
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
    <asp:ListView ID="lsvKlanten"
runat="server" ClientIDRowSuffix="Key">
        <ItemTemplate>
        <asp:TextBox ID="txtNieuweNaam"
runat="server"

ClientIDMode="Predictable"></asp:TextBox>
        </ItemTemplate>
    </asp:ListView>
    </form>
</body>
</html>
```

CODEVOORBEELD 4.

```
<%@ Page Language="C#"
ClientIDMode="Static"
AutoEventWireup="true"
MasterPageFile="~/MasterPage.master"
CodeFile="Default.aspx.cs" Inherits="_De-
fault" %>
```

CODEVOORBEELD 5.

ViewState

Een van de belangrijkste punten van aandacht bij het ontwerpen en implementeren van een ASP.NET applicatie is de viewstate. Deze viewstate is niet direct zichtbaar op de HTML-pagina, maar zal als een hidden element worden gerendered in de HTML-pagina. Wanneer een pagina veel webcontrols bevat, kan deze viewstate enorm groot worden met als gevolg dat de performance van de pagina negatief zal worden beïnvloed. Daartoe is er in de oudere versie van ASP.NET een mogelijkheid gebouwd om de viewstate voor bepaalde controls uit te zetten. Echter, wanneer een parent control de viewstate uit had staan, ging voor de child controls ook automatisch de viewstate uit. Bijvoorbeeld:

```
<asp:Panel ID="pnlUpdate" runat="server"
EnableViewState="false">
  <asp:Label ID="txtLabel"
runat="server"></asp:Label>
</asp:Panel>
```

Dit zal tot gevolg hebben dat voor het label txtlabel ook de viewstate uit zal staan. Dit is in veel gevallen niet wenselijk en geeft niet de volledige controle over wat wel en niet in de viewstate moet komen. Daartoe is er een property met de naam "ViewStateMode" toegevoegd. Deze property kan de volgende drie waarden hebben:

- Disabled. De viewstate voor het control zal uit staan.
- Inherit. De viewstate voor het control is gelijk aan de parent.
- Enabled. De viewstate zal expliciet worden aangezet.

Wanneer dus de volgende code wordt gebruikt, zal de viewstate voor txtlabel aan staan, terwijl de parent deze op disabled heeft staan.

```
<asp:Panel ID="pnlUpdate" runat="server"
ViewStateMode="Disabled">
  <asp:Label ID="txtLabel" runat="server"
ViewStateMode="Enabled"></asp:Label>
 </asp:Panel>
```

URL Routing

URL Rewriting is altijd een populaire techniek geweest om Search Engine optimalisation te realiseren. Search Engines kunnen URLs als http://www.test.nl/productcategories.aspx?rootcategory=computers niet goed indexeren. Daarnaast is het zo dat dit soort URLs moeilijk zijn te onthouden. Om deze URL SEO-friendly te maken, is een URL als http://www.test.nl/productcategories/computers al beter. Om nu deze URL naar een specifieke pagina binnen de site te laten wijzen, maakt men gebruik van URL Rewriting. Tot ASP.NET 3.5 sp1 werd daar meestal een los component voor gebruikt. Sinds ASP.NET 3.5 SP1 is er binnen ASP.NET echter URL Routing beschikbaar om dit te realiseren. In ASP.NET 4.0 is deze techniek nog verder uitgebreid en vervolmaakt. Om URL Routing te kunnen realiseren zal allereerst binnen de application een RoutingTable moeten worden opgezet (zie codevoorbeeld 6).

```
void Application_Start(object sender,
EventArgs e)
    {

RouteTable.Routes.Add("TestRoute", new
Route("test/{catnaam}/{produktnaam}", new
PageRouteHandler("~/categories.aspx")));
    }
```

CODEVOORBEELD 6.

Op dit moment zullen alle requests die binnenkomen, worden gematchd met deze tabel. Wanneer de URL er bijvoorbeeld als volgt uitziet : http://www.test.nl/test/Category1/productA, dan zal deze URL worden afgehandeld door het URL Routing-mechanisme. Dit heeft tot gevolg dat de pagina 'categories.aspx' zal worden uitgevoerd. Binnen deze pagina zijn vervolgens de variabelen te gebruiken die in de URL zijn opgenomen. In dit voorbeeld is dat 'categorienaam' en 'productnaam'. Deze variabelen worden echter niet via het standaard request object beschikbaar gesteld, maar via het RoutData-object. Dit object is onderdeel van de RequestContext, maar kan ook via Page.RoutData worden benaderd. Codevoorbeeld 7 laat zien hoe je de waarden van de variabelen kunt gebruiken.

```
protected void Page_Load(object sender,
EventArgs e)
    {
        Response.Write(Page.RouteData.
Values["catnaam"].ToString());
        Response.Write(Page.RouteData.
Values["produktnaam"].ToString());
    }
```

CODEVOORBEELD 7.

Wanneer je URL Routing binnen een pagina gebruikt, moet dit ook door worden gevoerd binnen de links op de pagina zelf. Stel dus dat je een productpagina laat zien en op deze pagina links wilt laten zien naar andere producten. Bij URL Rewriting zou dit betekenen dat je via de codebehind de link zou moeten samenstellen. Binnen ASP.NET 4.0 is het mogelijk om hiervoor een expressie in de markup van de pagina te gebruiken:

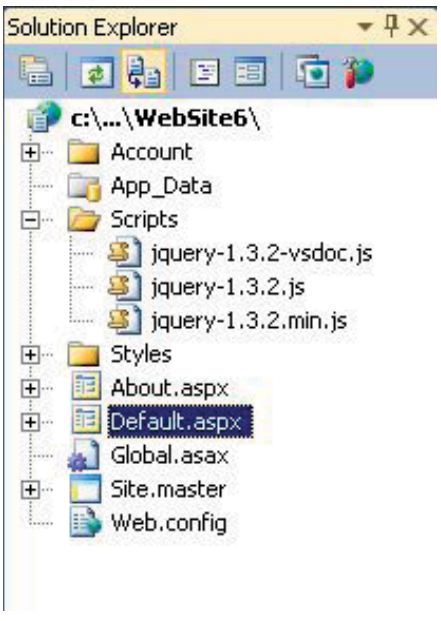
```
<asp:HyperLink ID="produktLink"
runat="server"
NavigateUrl="<%$RouteUrl:CatNaam=Category2
,Produktnaam=ProductB%>">
Details Product B</asp:HyperLink>
```

In dit voorbeeld zal de ASP.NET runtime de URL gebruiken zoals deze is gebruikt tijdens het URL Routing-proces. Vervolgens zullen de variabelen in deze URL worden vervangen door de waarden zoals opgegeven in de markup. Wanneer de variabele niet voorkomt in de URL, zal deze als querystring worden toegevoegd aan de URL. Voordeel hiervan is dus dat, wanneer de URL zal wijzigen, je alleen de routingtable aan hoeft te passen (als tenminste de variabelen gelijk blijven).

Microsoft Ajax Library

De Microsoft Ajax Library is geen standaard onderdeel van ASP.NET 4.0, maar is een aparte download. De reden hiervoor is dat deze Library open source is. Er zijn echter een aantal ontwikkelingen in deze Library die zeer de moeite waarde zijn om te noemen. Zo kan met een aantal nieuwe componenten in deze Library optimaal gebruik worden gemaakt van het laden van Javascript-files en zijn er componenten die het gebruik van data vanuit WCF-services zeer vereenvoudigen. Ook is er nu een naadloze integratie met JQuery. Zoals wellicht bekend is de JQuery Library inmiddels uitgegroeid tot een van de meest populaire Javascript Libraries. Dit heeft met name te maken met het eenvoudige gebruik van de Library, alsmede met de grote hoeveelheid aan componenten die hiervoor beschikbaar is. Ook Microsoft ziet deze ontwikkeling en heeft JQuery als standaard component opgenomen in Visual Studio 2010. Wanneer je een nieuw project aanmaakt,zul je in de scripts directory de JQuery-files vinden (Afbeelding 2). In deze directory zul je 3 JQuery-files vinden. De file 'jquery-1.3.2.js' is de standaard JQuery-file. De file 'jquery-1.3.2.min.js' is dezelfde file, maar dan minimized. Dat wil dus zeggen dat alle overbodige spaties, etc. eruit zijn gehaald.
Dit heeft tot gevolg dat de bestandsgrootte van dit bestand minimaal is en dus is geoptimaliseerd om te downloaden vanaf een website. De file 'jquery-1.3.2-vsdoc.js' bevat XML Comments die door de intellisense zullen worden gebruikt binnen de Visual Studio 2010-omgeving.

Zoals aangegeven bevat de Ajax Library controls om het downloaden van de gebruikte Javascript-files te optimaliseren. Het control ScriptLoader zorgt hiervoor. Standaard weet dit control alles van de controls die in de Ajax Library beschikbaar zijn. Het is echter ook mogelijk om binnen je eigen scripts gebruik te maken



AFBEELDING 2.

van dit control. Om te beginnen moet er een referentie worden gemaakt naar het bestand start.js. Dit bestand is onderdeel van de Ajax Library en heeft alle informatie in zich over de verschillende controls binnen de Library. Zo weet het control precies voor welke controls welke .js-files nodig zijn. Dit bestand is echter ook nodig om onze eigen scripts toe te voegen. Om een eigen scriptfile toe te voegen, kan gebruik worden gemaakt van Codevoorbeeld 8. Deze code staat in een bestand met de naam CustomScripts.js.

```
Sys.loader.defineScripts({
    releaseUrl: "Scripts/{0}.js",
    debugUrl: "Scripts/{0}.js"
},
        [
            { name: "CustomScripts",
                executionDependencies:
["ApplicationServices"],

isLoaded: !!(window.My && My.Scripts &&

My.Scripts.TheScript)
            }
        ]
    );
```

CODEVOORBEELD 8.

Hier zie je dat we aan de loader een release- en debugscript toevoegen. De naam van het script moet de naam van de file zijn, zonder de extensie .js. Vervolgens kun je in de ASPX-file refereren naar deze file, middels codevoorbeeld 9.

```
<head id="Head1" runat="server">
    <title>Demo Application</title>
<script src="Scripts/MicrosoftAjax/start.
js"
            type="text/javascript"/>

<script src="Scripts/RegisterScripts.js"
type="text/javascript"/>
</head>
```

CODEVOORBEELD 9.

Dit zijn de enige twee referenties in de aspx-pagina. Deze bestanden zullen dus worden geladen op het moment dat de ASPX-pagina wordt geopend. De code in codevoorbeeld 10 is de inhoud van het bestand CustomScripts.js. Hierbij ishet belangrijk om op te merken dat we alleen de RegisterScript-method gebruikt, op het moment dat de scriptloader beschikbaar
is. Dit is gedaan om compatibiliteit te garanderen met webapplicaties die nog geen gebruikmaken van de Microsoft Ajax Library.

```
function () {
    function execute() {

clickButton=function(){ alert("Done
Click") };
    }

    if (window.Sys && Sys.loader) {
        Sys.loader.
registerScript("CustomScripts", null, exe-
cute);
    }
    else {
        execute();
    }
})();
```

CODEVOORBEELD 10.

Alles is nu gereed om methods aan te roepen in het bestand CustomScripts.js. Zoals eerder opgemerkt, is het bestand dus niet geladen binnen de ASPX-pagina. Het bestand zal pas worden geladen wanneer een method wordt gebruikt die het bestand CustomScripts.js nodig heeft. Dit is te zien in codevoorbeeld 11.

```
<script type="text/javascript">
     function ClickHandler() {

Sys.require(Sys.scripts.CustomScripts,
function () {
            clickButton();
        });
    }
</script>
<input type="button"
onclick="ClickHandler()" value="Klik" />
```

CODEVOORBEELD 11.

Op het moment dat de methode ClickHandler wordt aangeroepen, zal daarbinnen de function clickButton worden aangeroepen. Echter, voordat dit gebeurt, wordt gecontroleerd of de file, die gedefinieerd is in Sys.scripts.CustomScripts, is geladen. Zo niet, dan zal deze dynamisch worden geladen. Dit is vanuit het oogpunt van performance natuurlijk ideaal aangezien de .js-file pas geladen wordt op het moment dat deze nodig is. Wanneer een bepaalde knop op de pagina niet gebruikt wordt, zullen de .js-files dus ook niet worden geladen.
Daarnaast zijn er een groot aantal wijzigingen doorgevoerd in de Ajax Library die het mogelijk maken om data vanaf de server (webservices) te halen en via Javascript te benaderen. De toevoegingen die daarbij in het oog springen, zijn de AdoNetServiceproxy-, DataContext- en de ADONetDataContext-classes. Deze classes zorgen ervoor dat het vanuit de client eenvoudig is om te communiceren met ADO.NET Data Services (AdoNetServiceProxy) en webservices (DataContext en ADONetDataContext). Als voorbeeld gaan we data van een WCF-service gebruiken middels de Ajax Client Library. Allereerst moet er een Ajax Enabled WCF service toegevoegd worden. Zie codevoorbeeld 12.

```
[ServiceContract(Namespace = "Qurius")]
[AspNetCompatibilityRequirements(Requireme
ntsMode = AspNetCompatibilityRequirements-
Mode.Allowed)]
public class Service
{
    public class CustomerData
    {
        public string Name { get; set; }
        public string Code { get; set; }
    }

    [OperationContract]
    [WebGet()]
    public List<CustomerData> Return-
Clients()
    {
        return new List<CustomerData>()  {

new CustomerData() { Name = "Klant1", Code
= "Code1" },

new CustomerData() { Name = "Klant2", Code
= "Code2" }
        };
    }
}
```

CODEVOORBEELD 12.

Nu is deze WCF-service beschikbaar voor de Microsoft Ajax Library. Door de het Sys.Data.DataContext-object te combineren met de Sys.UI.DataView, kun je met een paar simpele regels Javascript een call doen naar de WCF-service en op de client een databind doen aan deze service. Zie codevoorbeeld 13.

```
function loadData() {
    var context = $create(Sys.Data.Data-
Context,

{ serviceUri: "Service.svc" });

var customersTemplate = $create(Sys.UI.
DataView, {
                autoFetch: true,
                dataProvider: context,
                httpVerb: "GET",
                fetchOperation: "Return-
Clients"},
           null,
           null,
           $get("customers-template"));
}
```

CODEVOORBEELD 13

De uiteindelijke databind zal plaatsvinden in de table met het ID customers-template.
Kortom, het is veel eenvoudiger geworden om data op de client te binden aan data die middels een WCF-service beschikbaar zijn gesteld.

Er is nog veel meer…

Naast alle verbeteringen die zijn genoemd in dit artikel zijn er nog een groot aantal wijzigingen in het ASP.NET framework doorgevoerd, waardoor dit framework nog interessanter wordt om te gebruiken voor webapplicaties. Zo is bijvoorbeeld het gehele cachingmechanisme binnen ASP.NET gewijzigd naar een providermodel. Deze is nu zo ontworpen dat je zelf providers kunt schrijven die de handling van de cache uitvoeren.
Je kunt zelfs verschillende cacheproviders toevoegen, zodat je per pagina aan kunt geven welke cachingprovider gebruikt moet worden. Dit opent oa de wegen naar een distribute caching-mechanisme (caching over verschillende caching servers).
Ook zijn er nog wijzigingen doorgevoerd in de dynamic data classes en in het session state management. Een overgang naar ASP.NET 4.0 en een combinatie met VS2010 is meer dan de moeite waard!

Links

Webdeployment-opties binnen Visual Studio 2010:
http://vishaljoshi.blogspot.com/2009/09/overview-post-for-web-deployment-in-vs.html
ASP.NET componenten (o.a. Microsoft Ajax Library en MVC)
http://aspnet.codeplex.com/
Scott Guthrie (.NET 4.0 en VS 2010)
http://weblogs.asp.net/Scottgu/

Patrick Smits, is Software Architect bij Qurius Advanced Solutions. Hij is te bereiken via email patrick.smits@qurius.com.