Bart Reyserhove

Castle Monorail in de praktijk

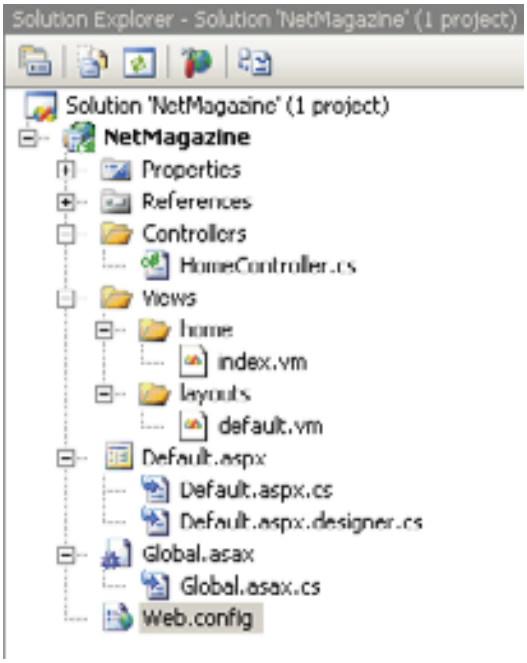
EEN ANDERE KIJK OP HET BOUWEN VAN EEN WEBAPPLICATIE

Webforms is al geruime tijd de standaard als het gaat om het bouwen van webapplicaties. Dit wil echter niet zeggen dat er geen alternatieven zijn. In dit artikel bekijken we hoe we met behulp van Monorail op een andere manier webapplicaties kunnen bouwen. Belangrijk om vanaf het begin te weten is dat Monorail bovenop ASP.NET draait en alleen webforms vervangt.

Monorail is een web application framework dat gebruikmaakt van het model-view-controller-patroon. Het *Model* bevat de businessobjecten. In het geval van Monorail kun je hier gebruikmaken van ActiveRecord, een object relational mapper (ORM) die bovenop NHibernate is gebouwd. Je bent echter niet verplicht om ActiveRecord te gebruiken, je kunt ook een andere ORM gebruiken of met ADO.NET werken. De *Controller* is uitsluitend verantwoordelijk voor de applicatieflow en de *View* is alleen begaan met de presentatielogica. Verder maakt Monorail ook gebruik van een aantal principes uit Ruby On Rails. Convention over configuration is daar het belangrijkste van. Dit wil zeggen dat je een aantal afspraken dient te kennen en dat je dan bijna niets hoeft te configureren. Wijk je af van de afspraken, dan moet je extra code te schrijven.

Een blik onder de motorkap

Monorail implementeert het 'Front Controller'-patroon door gebruik te maken van een eigen HttpHandler. Wanneer er een request (bijvoorbeeld voor home/index.rails) binnenkomt, zal Monorail de url ontleden om de correcte methode aan te roepen in de controller. In het geval van 'home/index.rails' zal dit de indexmethode zijn in de 'home' controller. De indexmethode is verantwoordelijk voor de verdere verwerking van de request en zal daarna ook de view aanroepen die gebruikt wordt om de webpagina weer te geven. Aangezien Monorail bovenop ASP.NET draait, zal het ook voordeel halen uit de features ervan zoals Session Management, Authentication/Authorization provider.



Afbeelding 1. Projectstructuur in Solution Explorer

Het model

Hoog tijd dat we overgaan naar de praktijk. Om te beginnen dien je Monorail te downloaden. Nadat je het hebt geïnstalleerd, zal je zien dat bij het begin van een nieuw project in Visual Studio er een nieuwe template beschikbaar is waarmee je een Monorail-project kunt starten. De projectstructuur ziet er dan uit zoals in afbeelding 1. Vervolgens beginnen we met code te schrijven voor het *Model*. Laten we voor het gemak even uitgaan van één tabel in onze database 'Wines'. De code van het model kun je bekijken in codevoorbeeld 1. Je ziet dat de klasse 'Wine' overerft van ActiveRecordValidationBase. Dit zal ons toelaten om databaseoperaties te verrichten op het object Wine. Verder zie je ook dat er attributen toegevoegd zijn aan de verschillende properties om ze te mappen naar kolommen in de database. Omdat we de klasse Wine hebben laten overerven van ActiveRecordValidationBase kunnen we ook onmiddellijk validatieregels op de properties plaatsen. In codevoorbeeld 1 zie je bijvoorbeeld dat 'Naam' een verplicht veld is. Later, in de code van de controller, zien we hoe je deze regels afdwingt. Nu is onze model klaar.

Scaffolding

Het model dat we juist hebben aangemaakt, gaan we nu gebruiken in de *Controller*. Daarvoor maken we een klasse aan die we WineController noemen. Met behulp van de scaffolding-functionaliteit van Monorail kunnen we nu automatisch webpagina's genereren voor ons Wine-object. Scaffolding is ontleend aan Ruby On Rails, en wil zeggen dat je de compiler het model laat gebruiken om code te genereren, die je toelaat om de basis databaseoperaties op een object uit te voeren (create, read, update, delete). In codevoorbeeld 2 zie je hoe je dat kunt doen in Monorail. Als je nu met je browser naar 'wine/list.rails' navigeert, krijg je de lijst van wijnen in de database te zien. Die zal in eerste instantie leeg zijn, maar de functionaliteit om wijnen toe te voegen, te editeren en te verwijderen is automatisch aanwezig. Door gebruik te maken van de scaffolding-functionaliteit kun je op een zeer snelle manier een eenvoudige webapplicatie bouwen, waarmee je een aantal testen kunt doen. Vanaf het moment dat je in de controller voor een bepaalde actie zelf code begint te schrijven, zal het framework die code gaan gebruiken en niet meer de standaard scaffoldingcode. Op die manier kun je zeer geleidelijk functionaliteit toevoegen aan jouw applicatie. Het is natuurlijk ook mogelijk om op de gewone manier te werken en hoe dat gaat, laten we zien in de volgende paragraaf.

```
using Castle.ActiveRecord;
using Castle.Components.Validator;
namespace WineCellar.Data
{
  [ActiveRecord]
  public class Wine : ActiveRecordValidationBase<Wine>
  {
    #region Private members
    private string name;
    private string appelation;
    private int id;
    #endregion

    #region Public properties
      [Property("Name", ColumnType = "String", NotNull = true),
    ValidateNonEmpty("Naam is een verplicht veld.")]
    public string Name
    {
      get{return this.name;}
      set{this.name = value;}
    }
    [Property("Appelation", ColumnType = "String")]
    public string Appelation
    {
      get{return this.appelation;}
      set{this.appelation = value;}
    }
    [PrimaryKey(PrimaryKeyType.Identity)]
    public int Id
    {
      get{return this.id;}
      set{this.id = value;}
    }
    #endregion
  }
}
```

Codevoorbeeld 1.

```
using Castle.MonoRail.Framework;
using WineCellar.Data;

namespace NetMagazine.Controllers
{
  [Scaffolding(typeof(Wine))]
  public class WineController :Controller
  {
  }
}
```

Codevoorbeeld 2.

Werken met data

We werken verder met dezelfde klasse (WineController) maar nu verwijderen we het Scaffolding-attribuut boven aan de klasse. In plaats van de klasse te laten overerven van 'Controller', erft ze nu over van 'SmartDispatcherController'. Dit zal ons toelaten om elementen van een webform te binden aan argumenten van een methode in de controller. Codevoorbeeld 3 is een *View* om een nieuwe wijn aan te maken. Bij het submitten van de form naar de controller willen we het wine-object opvullen. Om dat te bereiken, moeten de input-elementen in de view gebruikmaken van een prefix. In ons voorbeeld is dat de prefix wine. In de *Controller*-methode (zie codevoorbeeld 4) dienen we dan diezelfde prefix als argument te specificeren voor het 'DataBind'-attribuut. Daardoor zal het framework het wine-object instantiëren en opvullen. In de tweede parameter van het DataBind-attribuut specificeren we dat Monorail het wine-object dient te valideren . (Validate = true). De validatie gebeurt op basis van de validatieregels in het model; zie codevoorbeeld 1. Door middel van de methode HasValidationEr-

```
<h3>Create new wine</h3>
#if ($summary)
  <p>
    #foreach ($errmessage in $summary.ErrorMessages)
    $errmessage <br/>
    #end
  </p>
#end
<form action="create.rails" method="post">
  <div>
    <p>
      <label>Name</label>
      <input type="text" name="wine.name" />
    </p>
    <p>
      <label>Appelation</label>
      <input type="text" name="wine.appelation" />
    </p>
  </div>
  <p>
    <input type="submit" value="Create" />
  </p>
</form>
```

Codevoorbeeld 3.

```
using Castle.MonoRail.Framework;
using WineCellar.Data;

namespace NetMagazine.Controllers
{
  //[Scaffolding(typeof(Wine))]
  public class WineController :SmartDispatcherController
  {
    public void New()
    {
    }
    public void Create([DataBind("wine",Validate = true)] Wine wine)
    {
      if (!HasValidationError(wine))
      {
        wine.CreateAndFlush();
        RedirectToAction("list");
      }
      else
      {
        Flash["summary"] = GetErrorSummary(wine);
        RedirectToReferrer();
      }
    }
    public void List()
    {
      PropertyBag["wines"] = Wine.FindAll();
    }
  }
}
```

Codevoorbeeld 4.

ror kunnen we dan verifiëren of er fouten zijn opgetreden en een samenvatting van de fouten opslaan in een Flash. In codevoorbeeld 3 zie je hoe we die fouten kunnen tonen in de browser. Met Monorail is het ook mogelijk client-side validatie toe te passen op basis van de server-side-attributen in het model. Als er geen fouten zijn opgetreden bij de validatie kunnen we het wine-object opslaan in de database en de gebruiker doorverwijzen naar de lijst met wijnen. Dit kunnen we doen door Wine.CreateAndFlush() aan te roepen en dan RedirectToAction('list'). In de methode List()van de controller halen we dan alle wijnen terug uit de database door middel van Wine.FindAll() en we slaan ze op in een PropertyBag.

```
<h3>Lijst van wijnen</h3>
<table width="100%" border="1" cellpadding="2" cellspacing="0">
<tr>
   <th>Nr.</th>
      <th>Naam</th>
      <th>Appelation</th>
   </tr>
   #foreach ($wine in $wines)
   <tr>
      <td>$velocityCount</td>
      <td>$!{wine.name}</td>
      <td>$!{wine.appelation}</td>
   </tr>
   #end
</table>
<a href="../wine/new.rails">Nieuwe wijn</a>
```

Codevoorbeeld 5.

De PropertyBag wordt door Monorail gebruikt om variabelen naar de View door te geven. In de View kunnen we deze dan aanroepen met behulp van het $-teken; zie codevoorbeeld 5.

Troeven

Dit artikel beschrijft maar een zeer klein gedeelte van de mogelijkheden die Monorail biedt en het zou te ver leiden om alle mogelijkheden op te sommen. Monorail beschikt ook over functionaliteit om masterpages in te bouwen en om met user-controls te werken. Je kunt ook zeer gemakkelijk meertaligheid inbouwen of aantrekkelijke user-interfaces bouwen met Ajax. Monorail beschikt kortom over een groot aantal troeven waardoor het leven van een webontwikkelaar veel gemakkelijker wordt. Ik raad aan om zeker de referenties eens te checken om te ontdekken wat voor een boeiende gemeenschap er zich achter dit open source framework bevindt.

Bart Reyserhove is senior consultant bij Capgemini (http://www.be.capgemini.com). Hij is te bereiken via bart.reyserhove@capgemini.com

Referenties

Front Controller Pattern: http://msdn2.microsoft.com/en-us/library/ms978723.aspx
Download Castle Monorail: http://www.castleproject.org/castle/download.html
Wiki met tal van voorbeelden: http://using.castleproject.org/dashboard.action
Blog van de stichter van Castle Project: http://hammett.castleproject.org/
Blog met tal van Monorail posts: http://www.ayende.com/Blog/
Een eenvoudig wijnkelderbeheerssysteem gebaseerd op Monorail: http://code.google.com/p/winecellarmanager/
Ajax Scaffolding met Monorail: http://danbunea.blogspot.com/2006/12/ajax-scaffolding-with-castle-monorail.html