

# Modeller, datainmatning och validering



# Upphovsrätt för detta verk

Detta verk är framtaget i anslutning till kursen ASP.NET MVC vid Linnéuniversitetet.

## Du får använda detta verk så här:

Allt innehåll i detta verk av Mats Looch, förutom Linnéuniversitetets logotyp och symbol samt ikoner och fotografier, är licensierad under:



Creative Commons Erkännande-IckeKommersiell-DelaLika 2.5 Sverige licens.

<http://creativecommons.org/licenses/by-nc-sa/2.5/se/>

## Det betyder att du i icke-kommersiella syften får:

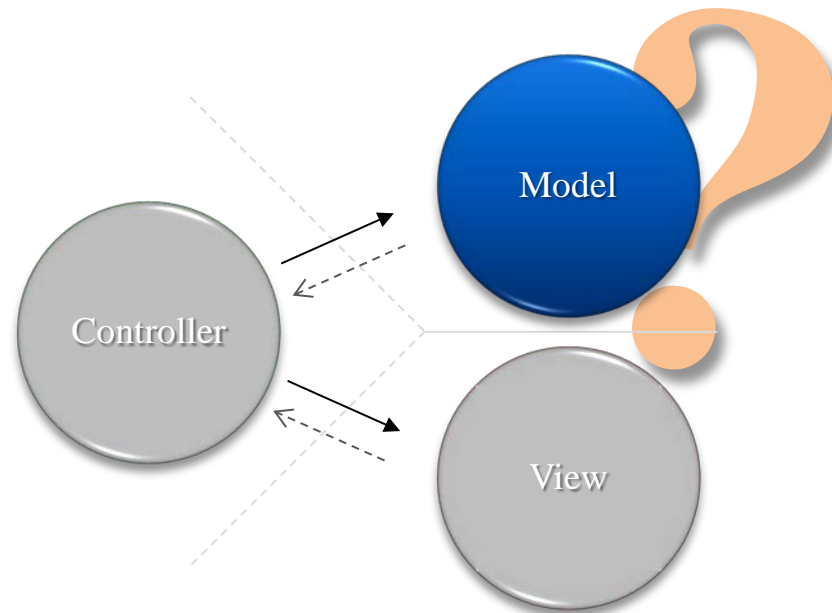
- kopiera hela eller delar av innehållet
- sprida hela eller delar av innehållet
- visa hela eller delar av innehållet offentligt och digitalt
- konvertera innehållet till annat format
- du får även göra om innehållet

Om du förändrar innehållet så ta inte med Linnéuniversitetets logotyp och symbol samt ikoner och fotografier i din nya version!

Vid all användning måste du ange källan: "Linnéuniversitetet – ASP.NET MVC" och en länk till <https://coursepress.lnu.se/kurs/aspnet-mvc> och till Creative Common-licensen här ovan.

# Vad är en modell?

- ✓ I MVC används modell för att beteckna dataobjektet som skickas från kontrollern till vyn.
- ✓ ASP.NET MVC har inga krav på att en typ som representerar en modell ska ärva från en specifik basklass.
- ✓ En modell har i regel publika egenskaper.
  - En modells egenskaper kan renderas av vyn genom att använda *HTML Helpers*.
  - Formulärdata kan automatiskt av en controller bindas till en modells egenskaper.
  - Genom att dekorera en egenskap med attribut kan validering implementeras på server och klient.



# En enkel klass är modellen

- ✓ En instans av klassen Birthday, en helt vanlig klass, är ett exempel på en modell som kan skickas från en controller till en vy.
- ✓ Egenskaperna Birthdate och Name används för initiera objektet.
- ✓ Age, DaysUntilNextBirthday och NextBirthdayDate är *read-only*-egenskaper vars värden är beroende av Birthdate.

```
public class Birthday
{
    public DateTime Birthdate { get; set; }
    public string Name { get; set; }
    public int Age
    {
        get
        {
            return this.NextBirthdayDate.Year - this.Birthdate.Year;
        }
    }
    public int DaysUntilNextBirthday
    {
        get
        {
            return (this.NextBirthdayDate - DateTime.Today).Days;
        }
    }
    public DateTime NextBirthdayDate
    {
        get
        {
            var nextBirthday = new DateTime(DateTime.Today.Year,
                this.Birthdate.Month, this.Birthdate.Day);
            if (nextBirthday < DateTime.Today)
            {
                nextBirthday = nextBirthday.AddYears(1);
            }
            return nextBirthday;
        }
    }
}
```

Birthday används till att bestämma antalet dagar det är till en persons nästa födelsedag.

# Ett enkelt formulär för data till modellen...

✓ ...men det finns enklare och bättre sätt.

```
1  @{
2      Layout = null;
3  }
4  <!DOCTYPE html>
5  <html>
6  <head>
7      <title>Nästa födelsedag</title>
8  </head>
9  <body>
10     <h1>Nästa födelsedag</h1>
11     <form action="/" method="post">
12     <div>
13         <label for="namn">
14             Namn</label>:</div>
15     <div>
16         <input name="namn" type="text" />
17     </div>
18     <div>
19         <label for="fodelsedatum">
20             Födelsedatum</label>:</div>
21     <div>
22         <input name="fodelsedatum" type="text" />
23     </div>
24     <p>
25         <input type="submit" value="Skicka" /></p>
26     </form>
27 </body>
28 </html>
```

Nästa födelsedag

Namn:

Ellen Nu

Födelsedatum:

2010-01-01

```
[HttpPost]
[ActionName("Index")]
public ActionResult Index_POST()
{
    var model = new Birthday
    {
        Name = Request.Form["namn"],
        Birthdate = DateTime.Parse(Request.Form["fodelsedatum"]);
    };
    return View("UpcomingBirthday", model);
}
```

Ingen optimal lösning!

# Ett enklare formulär med *HTML Helpers*

- ✓ Med hjälp av *HTML Helpers* kan ett formulär genereras.
- ✓ En *HTML Helper* är typiskt en metod som returnerar en sträng och kan användas för att generera HTML-element som textfält, länkar och listrutor.

The image is a composite showing the relationship between C# code, HTML output, and a rendered web form.

**C# Code (Left):** A code snippet using `Html.BeginForm()` to generate a form. It uses `Html.LabelFor` and `Html.EditorFor` for the Name and Birthdate fields, and `Html.Submit` for the submit button. The code is as follows:

```
using (Html.BeginForm())
{
    <div>
        @Html.LabelFor(model => model.Name)
    </div>
    <div>
        @Html.EditorFor(model => model.Name)
    </div>

    <div>
        @Html.LabelFor(model => model.Birthdate)
    </div>
    <div>
        @Html.EditorFor(model => model.Birthdate)
    </div>

    <p>
        <input type="submit" value="Skicka" />
    </p>
}
```

**HTML Output (Bottom Left):** The corresponding HTML generated by the code:

```
<form action="/" method="post">
<div>
    <label for="Name">
        Name</label>
    </div>
<div>
    <input class="text-box single-line" id="Name" name="Name" type="text" value="" />
    </div>
<div>
    <label for="Birthdate">
        Birthdate</label>
    </div>
<div>
    <input class="text-box single-line" id="Birthdate" name="Birthdate" type="text" value="2010-01-01" />
    </div>
<div>
    <input type="submit" value="Skicka" />
    </div>
</form>
```

**Rendered Form (Right):** A screenshot of the web form titled "Nästa födelsedag". It shows the "Name" field with the value "Ellen Nu" and the "Birthdate" field with the value "2010-01-01". A "Skicka" button is at the bottom.

**Annotations:**

- An orange callout box points to the "Name" field with the text: "Oops! `Html.LabelFor` använder namnet på egenskapen." (Oops! `Html.LabelFor` uses the name of the property.)
- A blue callout box points to the `id` and `name` attributes in the HTML code with the text: "Attributen `id` och `name` sätts till egenskapens namn." (The attributes `id` and `name` are set to the property's name.)

# Metadata bestämmer vad Html.LabelFor renderar

- ✓ Genom att dekorera modellklassens egenskaper med metadataattributet DisplayName kan du bestämma vad Html.LabelFor ska rendera.

```
using System.ComponentModel;
namespace NextBirthday.Models
{
    public class Birthday
    {
        [DisplayName("Födelsedatum")]
        public DateTime Birthdate { get; set; }

        [DisplayName("Namn")]
        public string Name { get; set; }

        public int Age
        {
            get
            {
                return this.NextBirthdayDate.Year
            }
        }

        public int DaysUntilNextBirthday
        {
            get
            {
                return (this.NextBirthdayDate
            }
        }

        public DateTime NextBirthdayDate
        {
            get
            {
                return this.Birthdate.AddYears(1);
            }
        }
    }
}
```

## Nästa födelsedag

Namn:

Födelsedatum:

# Binda formulärdatat till parametrar

- ✓ Det går att hämta ut formulärdatat manuellt och tilldela modellens egenskaper värdena, men det är smidigare att använda bindningsmekanismen som finns.
- ✓ Formulärdata kan automatiskt bindas till parametrar i en metod.

```
<form action="/" method="post">
<div>
  <label for="Name">
    Namn</label></div>
<div>
  <input id="Name" name="Name" type="text" value="" />
<div>
  <label for="Birthdate">
    Födelsedatum</label></div>
<div>
  <input id="Birthdate" name="Birthdate" type="text" />
<div>
  <input type="submit" value="Skicka" /></div>
</form>
```

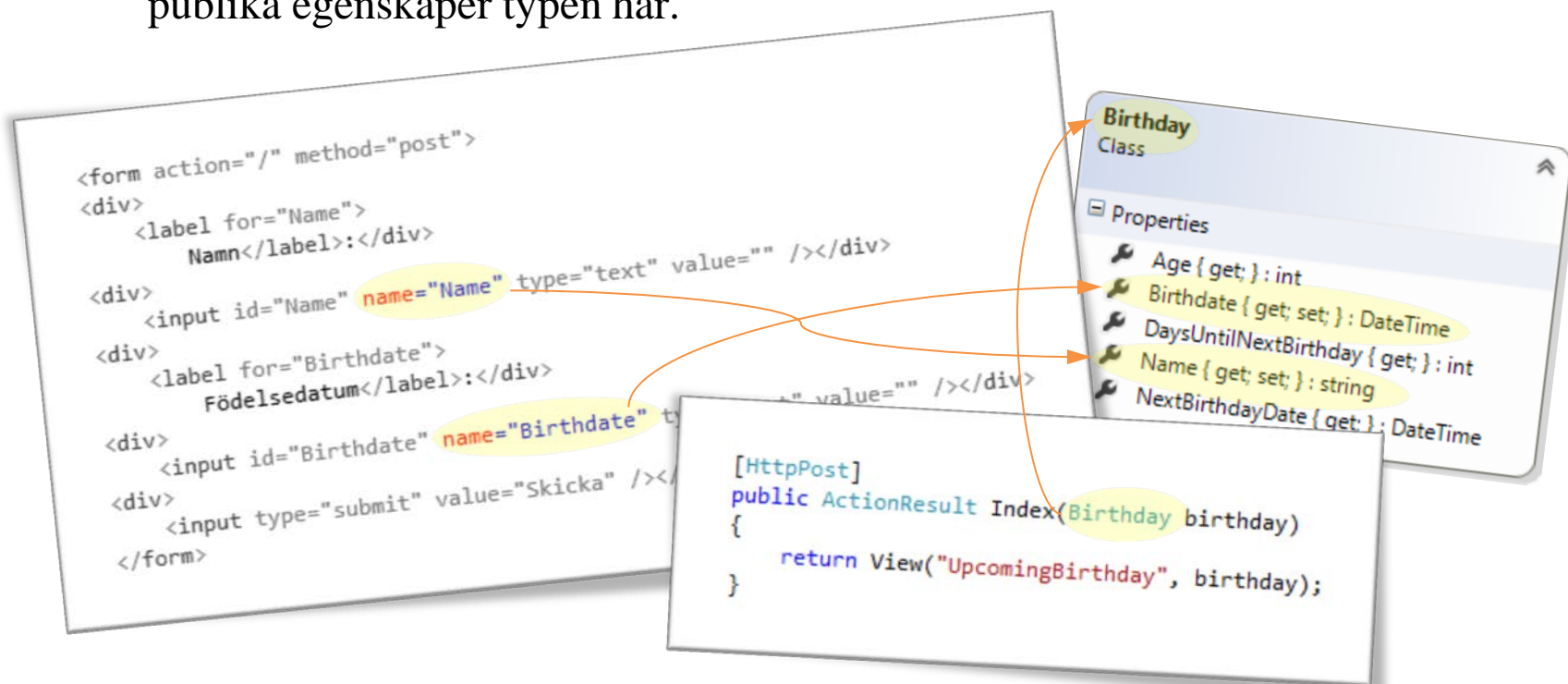
```
[HttpPost]
[ActionName("Index")]
public ActionResult Index_POST()
{
  var model = new Birthday
  {
    Name = Request.Form["namn"],
    Birthdate = DateTime.Parse(Request["birthdate"]);
  };
}
```

```
[HttpPost]
public ActionResult Index(string name, DateTime birthdate)
{
  var model = new Birthday
  {
    Name = name,
    Birthdate = birthdate
  };
  return View("UpcomingBirthday", model);
}
```



# Automatiskt binda formulärdata till en egen typ

- ✓ Formulärdata kan bindas till egna typer genom att `DefaultModelBinder`, komponenten som ansvarar för konverteringen av formulärdata, undersöker vilka publika egenskaper typen har.



# Binda data som har samma namn

- ✓ HTML-element med samma namn i ett formulär kan bindas till arrayer, samlingar eller associativa arrayer.

The diagram illustrates the process of binding multiple HTML elements with the same name to an array in an ASP.NET MVC application. It consists of several overlapping panels:

- HTML View:** A code snippet showing a form with 15 text boxes, all with the name attribute set to "wishlist".

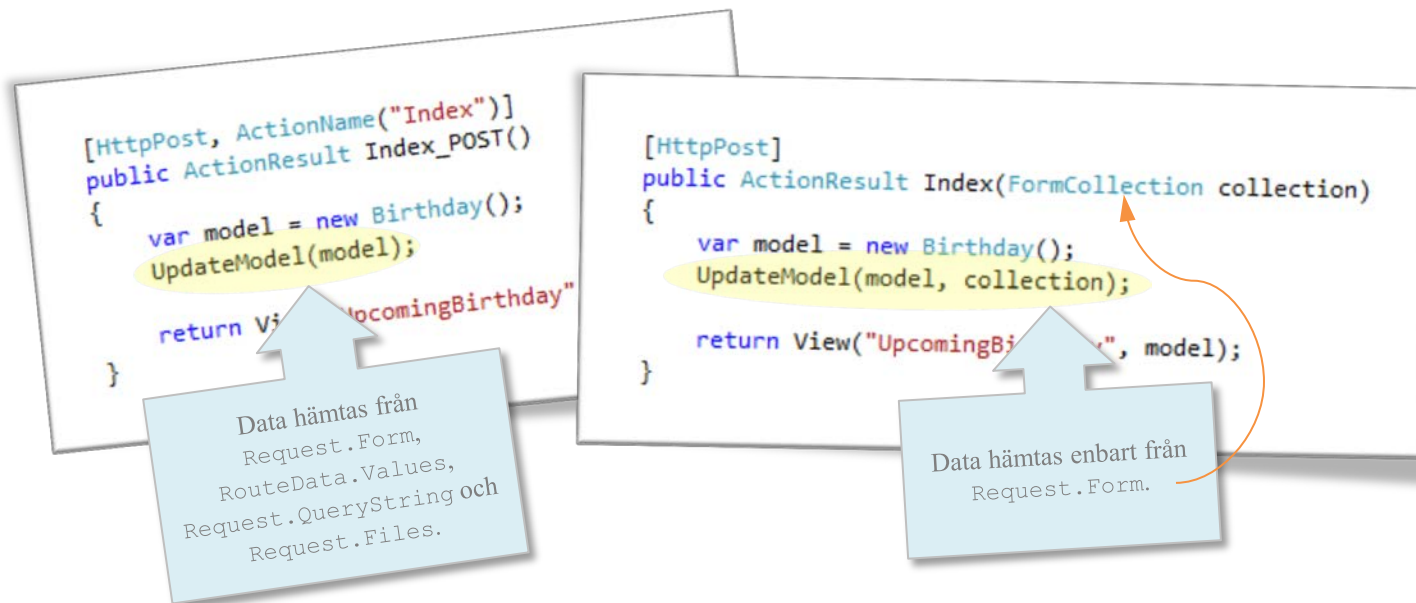
```
<h1> Önskelista</h1>
@using (Html.BeginForm())
{
    <div>
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
        @Html.TextBox("wishlist")
    </div>
    <p>
        <input type="submit" value="Skicka" /></p>
}
```
- Controller:** A code snippet showing a POST action method that receives the "wishlist" data as a string array.

```
[HttpPost]
public ActionResult Create(string[] wishlist)
{
    ...
}
```
- Model:** A diagram showing the data structure. It includes a variable "wishlist" of type "string[19]" and a collection of 10 items, each of type "string[19]".
- Form Data:** A list of items entered in the form, corresponding to the values in the model array: "Cykel", "Dator", "MP3", "Nintendo DS", "PC-spel", "Wii-spel", "PS2-spel", "Mobiltelefon", "Kläder", and "Böcker".

An orange arrow points from the "wishlist" attribute in the HTML code to the "wishlist" parameter in the controller method, indicating the binding process.

# Manuellt binda formulärdata till en egen typ

- ✓ Då du behöver större kontroll över hur modellobjektet instansieras kan du manuellt binda formulärdata till modellobjektet med metoderna `UpdateModel` och `TryUpdateModel`.
  - `UpdateModel` kastar ett undantag om bindningen misslyckas.
  - `TryUpdateModel` returnerar `false` om bindningen misslyckas.



# Modellens status

- ✓ MVC-ramverket använder egenskapen `ModelState`, en associativ array, för att lagra information om modellen. I `ModelState` lagras information som modellens status, bindningsfel, inkommande värden, ....

The image shows a screenshot of an ASP.NET MVC application. On the left, a form titled "Nästa födelsedag" (Next birthday) is displayed. It has two input fields: "Namn:" (Name) with the value "Ellen Nu" and "Födelsedatum:" (Birthdate) with the value "inget datum!" (no date). Both values are circled in red. Below the form, a snippet of C# code is visible, showing an `HttpPost` action method `Index` that takes a `FormCollection` and creates a `Birthday` model, updating it with the collection and rendering the view.

On the right, the Visual Studio "Watch" window is open, showing the state of the `ModelState` dictionary. The dictionary contains two entries:

- Key: `Count`, Value: `{System.Web.Mvc.ModelStateDictionary}`
- Key: `IsValid`, Value: `false`
- Key: `Count`, Value: `Count = 2`
- Key: `Count`, Value: `Count = 1`
- Key: `Errors`, Value: `{System.Web.Mvc.ModelError}`
- Key: `ErrorMessage`, Value: `"The value 'inget datum!' is not valid for Födelsedatum."`
- Key: `Raw View`, Value: `null`
- Key: `AttemptedValue`, Value: `{System.Web.Mvc.ValueProviderResult}`
- Key: `Culture`, Value: `"inget datum!"`
- Key: `RawValue`, Value: `{sv-SE}`
- Key: `Static members`, Value: `{string[1]}`
- Key: `Non-Public members`, Value: `{string[1]}`
- Key: `Errors`, Value: `{System.Web.Mvc.ModelError}`
- Key: `Value`, Value: `Count = 0`
- Key: `Culture`, Value: `{System.Web.Mvc.ValueProviderResult}`
- Key: `RawValue`, Value: `"Ellen Nu"`
- Key: `Static members`, Value: `{sv-SE}`

Orange arrows point from the "Ellen Nu" and "inget datum!" values in the form to the corresponding entries in the Watch window. A pink callout box with an arrow pointing to the error message in the Watch window contains the text: "Bindningsfel som orsakas av att strängen 'inget datum!' inte kan tolkas som ett datum." (Binding error caused by the string 'no date!' not being able to be interpreted as a date.)

# Felmeddelanden visas med *HTML Helper*

- ✓ Med hjälp av `Html.ValidationSummary` kan eventuella fel i modellen presenteras i ett `div`-element innehållande ett `ul`-element.
- ✓ Formulärfält som orsakar fel dekorerar MVC-ramverket med CSS-klassen `input-validation-error` oavsett om `Html.ValidationSummary` används eller inte.





# Lägga till egna felmeddelanden till ModelState

- ✓ Egna felmeddelanden läggs till ModelState med metoden AddModelError.
  - Om första parametern är namnet på ett formulärfält kopplas felmeddelandet till formulärfältet.

```
[HttpPost]
public ActionResult Index(FormCollection collection)
{
    var model = new Birthday();
    if (TryUpdateModel(model, new[] { "name", "birthdate" }, collection) &&
        model.Birthdate > DateTime.Today)
    {
        ModelState.AddModelError("Birthdate",
            "Datumet har inte infallit.");
    }

    if (ModelState.IsValid)
    {
        return View("UpcomingBirthday", model);
    }

    return View("Index", model);
}
```

## Nästa födelsedag

**Datumet har inte infallit.**

Namn:

Ellen Nu

Födelsedatum:

2018-04-26

Skicka

# Validering med *data annotation*

- ✓ Lämpligast sätt att validera är att använda *data annotation*. ASP.NET MVC känns vid sex attribut som kan användas av modellklassen, och det går att skapa egna.
  - [EmailAddress], [Range], [RegularExpression], [Required], [StringLength], [Url]
- ✓ Eventuella felmeddelanden läggs automatiskt till ModelState.

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;

namespace NextBirthday.Models
{
    public class Birthday
    {
        [Required(ErrorMessage = "Födelsedatum måste anges.")]
        [DataType(DataType.Date)]
        [DisplayName("Födelsedatum")]
        public DateTime Birthdate { get; set; }

        [Required(ErrorMessage = "Namn måste anges.")]
        [DisplayName("Namn")]
        public string Name { get; set; }

        public int Age
    }
}
```

Attributet `DataType` är inget valideringsattribut. Här ser det till att användargränssnittet undertrycker tiden och endast visar datumet.

## Nästa födelsedag

- Namn måste anges.
- Födelsedatum måste anges.

Namn:

Födelsedatum:

Skicka

# Placering av felmeddelanden

- ✓ Genom att använda `Html.ValidationMessageFor` kan felmeddelanden placeras var som helst i formuläret.

Värdet `true` ser till att fel knutna till ett fält inte visas i en osorterad lista.

```
@using (Html.BeginForm())
{
    @Html.ValidationSummary(true, "Fel inträffade. Korrigera det som är fel och försök igen.")
    <div>
        @Html.LabelFor(model => model.Name);
    </div>
    <div>
        @Html.EditorFor(model => model.Name)
        @Html.ValidationMessageFor(model => model.Name)
    </div>

    <div>
        @Html.LabelFor(model => model.Birthdate)
    </div>
    <div>
        @Html.EditorFor(model => model.Birthdate)
        @Html.ValidationMessageFor(model => model.Birthdate)
    </div>

    <p>
        <input type="submit" value="Skicka" /></p>
}
```

## Nästa födelsedag

Fel inträffade. Korrigera det som är fel och försök igen.

Namn:

Namn måste anges.

Födelsedatum:

Födelsedatum måste anges.

Skicka



# Validering på klienten

- ✓ ASP.NET MVC 4 använder sig av *data annotation* ihop med jQuery och jQuery Validation för validering på klienten ("*Unobtrusive Client Validation*").
- ✓ Valideringsmeddelanden visas och döljs dynamiskt och formuläret postas inte om fälten inte klarar valideringen på klienten.

```
<h1>Nästa födelsedag</h1>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    @Html.ValidationSummary(true, "Fel inträffade. Korrigera det s")
    <div class="editor-label">
        @Html.LabelFor(model => model.Name)
    </div>
    <div class="editor-field">
        @Html.EditorFor(model => model.Name)
        @Html.ValidationMessageFor(model => model.Name)
    </div>
    <div class="editor-label">
        @Html.LabelFor(model => model.Birthdate)
    </div>
    <div class="editor-field">
        @Html.EditorFor(model => model.Birthdate)
        @Html.ValidationMessageFor(model => model.Birthdate)
    </div>
    <p>
        <input type="submit" value="Skicka" />
    </p>
}
```

```
@Scripts.Render("~/bundles/jquery")
@Scripts.Render("~/bundles/jqueryval")
```

```
<div>
    <label for="Name">
        Namn</label>:</div>
    <div>
        <input class="text-box single-line valid" data-val="true" data-valmsg-for="Name" data-valmsg-generated="true" type="text" value="" />
        <span class="field-validation-valid" data-valmsg-for="Name" data-valmsg-generated="true" />
    </div>
    <div>
        <label for="Birthdate">
            Födelsedatum</label>:</div>
    <div>
        <input class="text-box single-line input-validation-error" data-val="true" data-valmsg-for="Birthdate" data-valmsg-generated="true" type="text" value="" />
        <span class="field-validation-error" data-valmsg-for="Birthdate" data-valmsg-generated="true" type="text" value="" />
    </div>
```

Namn:

Ellen Nu

Födelsedatum:

Födelsedatum mås

```
<configuration>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  </appSettings>
```

# Mer information

- ✓ I kurslitteraturen kapitel 20-23 hittar du mer information om ”*model templates*”, bindning, validering och ”*unobtrusive Ajax*”.
- ✓ På <http://www.asp.net/mvc/tutorials/mvc-4/getting-started-with-aspnet-mvc4/adding-validation-to-the-model> finns exempel på validering. På [http://www.asp.net/mvc/tutorials/older-versions/models-\(data\)/performing-simple-validation-cs](http://www.asp.net/mvc/tutorials/older-versions/models-(data)/performing-simple-validation-cs) finns flera ”gamla” exempel på hur validering kan implementeras.

