

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 Loock, 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:

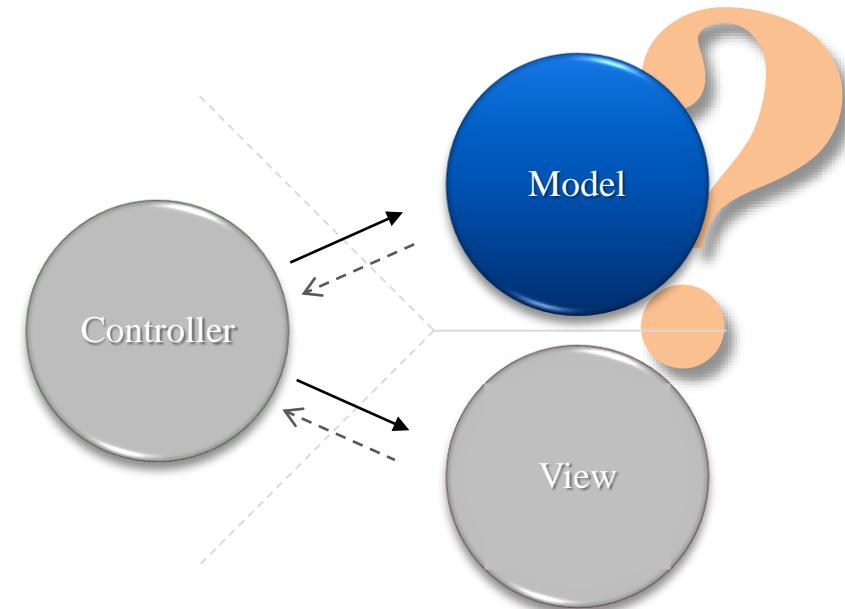
- kopiera hela eller delar av innehållet
- sprida hela eller delar av innehållet
- visa hela eller delar av innehållet offentligt och digitalt
- konvertera innehållet till annat format
- du får även göra om innehållet

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 controllern till vyn.
- ✓ ASP.NET MVC har inga krav på att en typ som representerar en modell ska ärvा 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
5  <!DOCTYPE html>
6
7  <html>
8  <head>
9      <meta name="viewport" content="width=device-width" />
10     <title>Nästa födelsedag (version 0)</title>
11 </head>
12 <body>
13     <h1>Nästa födelsedag (version 0)</h1>
14     <form action="/birthday/index" method="post">
15         <div>
16             <label for="namn">Namn</label>;
17             </div>
18             <div>
19                 <input name="namn" type="text" />
20             </div>
21             <div>
22                 <label for="fodesledatum">Födelsedatum</label>;
23             </div>
24             <div>
25                 <input name="fodesledatum" type="text" />
26             </div>
27             <div>
28                 <input type="submit" value="Skicka" />
29             </div>
30         </form>
31     </body>
32 </html>
```

Nästa födelseda

Namn: Ellen Nu

Födelsedatum: 2010-01-01

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

```
<form action="/birthday/index" method="post">
    <div>
        <label for="Name">Name:</label>
        <div>
            <@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>

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



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 - Birthdate.Year;
            }
        }

        public int DaysUntilNextBirthday
        {
            get
            {
                return (this.NextBirthdayDate - Birthdate).Days;
            }
        }

        public DateTime NextBirthdayDate
        {
            get
            {
                DateTime nextBirthday = new DateTime(DateTime.Today.Year,
                    Birthdate.Month, this.Birthdate.Day);
                nextBirthday = nextBirthday.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 binds automatiskt till parametrar i en metod.

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

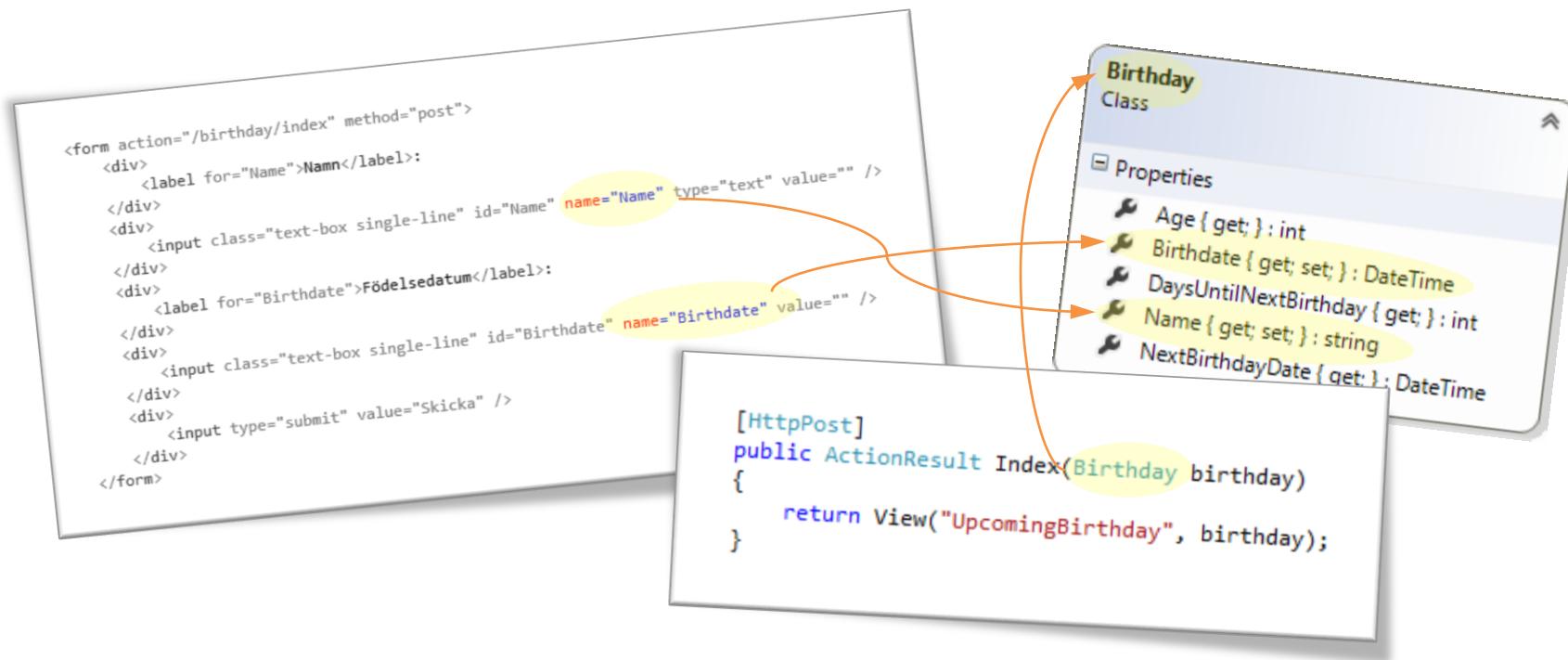
```
[HttpPost]
[ActionName("Index")]
public ActionResult Index_POST()
{
    var model = new Birthday
    {
        Name = Request.Form["namn"],
        Birthdate = DateTime.Parse(Request.Form["födelsedatum"]),
    };
    return View(model);
}
```

```
[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

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



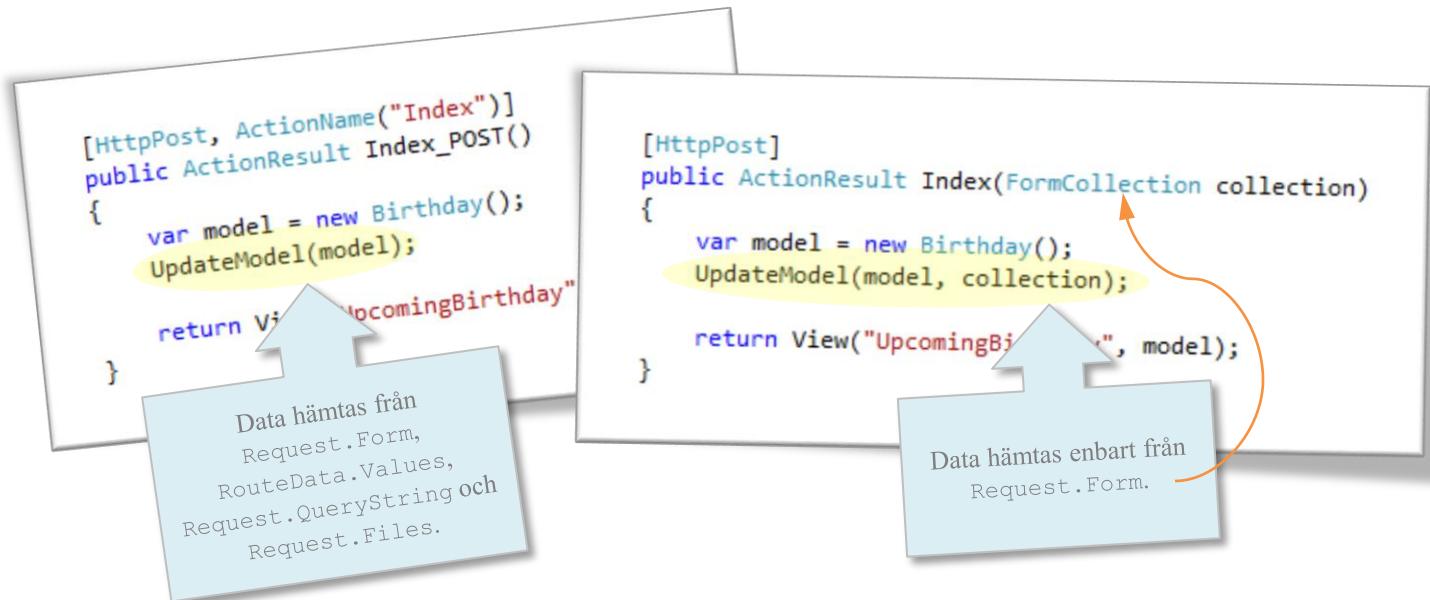
Binda data som har samma namn

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



Manuellt binda formulärdatat till en egen typ

- ✓ Då du behöver större kontroll över hur modellobjektet instansieras kan du manuellt binda formulärdatat 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 screenshot shows a Microsoft Visual Studio interface. On the left, there is a form with two fields: 'Namn:' containing 'Ellen Nu' and 'Födelsedatum:' containing 'inget datum!'. Both fields are circled in red. A large orange arrow points from these fields to the 'Watch' window on the right. The 'Watch' window displays the `ModelState` dictionary. It shows two entries: one for 'Namn' with a value of 'Ellen Nu' and another for 'Födelsedatum:' with a value of 'inget datum!'. The entry for 'Födelsedatum:' has an associated error message: "'The value 'inget datum!' is not valid for Födelsedatum.'". A callout bubble points to this message with the text: 'Bindningsfel som orsakas av att strängen "inget datum!" inte kan tolkas som ett datum.'

```
    [HttpPost]
    public ActionResult Index(FormCollection collection)
    {
        var model = new Birthday();
        UpdateModel(model, collection);
        return View("UncomingBirthDay", model);
    }
```

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 dekoreras av MVC-ramverket med CSS-klassen `input-validation-error` oavsett om `Html.ValidationSummary` används eller inte.

The screenshot shows a web page titled "Nästa födelsedag". A validation error message "The value 'inget datum!' is not valid for Födelsedatum." is displayed above the input field. The input field for "Födelsedatum" has a red border and contains the text "inget datum!". Below the input fields is a "Skicka" button.

```
for validation helpers
validation-error
color: #ff0000;
.field-validation-valid
{
    display: none;
}
.input-validation-error
{
    border: 1px solid #ff0000;
    background-color: #ffffcc;
}
.validation-summary
{
    font-weight: bold;
    color: #ff0000;
}
.validation-summary-error
{
    display: none;
}
```

The screenshot shows the generated HTML code for the "Nästa födelsedag" page. It includes the following code snippets:

```
<!DOCTYPE html>
<html name="viewport" content="width=device-width" />
<head>
    <title>Nästa födelsedag</title>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/css" rel="stylesheet" type="text/css" />
    <script src="~/bundles/modernizr" type="text/javascript" />
</head>
<body>
    <h1>Nästa födelsedag</h1>
    <@Html.BeginForm()>
        <@Html.AntiForgeryToken()>
        <@Html.ValidationSummary()>
        <@Html.LabelFor("Name")>
            Namn:<br/>
            <input class="text-box single-line" id="Name" name="Name" type="text" value="Ellen Nu" />
        </@Html.LabelFor("Name")>
        <@Html.LabelFor("Birthdate")>
            Födelsedatum:<br/>
            <input class="text-box single-line" id="Birthdate" name="Birthdate" type="text" value="inget datum!" />
        </@Html.LabelFor("Birthdate")>
        <p>Här saknas anrop till <code>Html.ValidationSummary</code>.</p>
        <input type="submit" value="Skicka" />
    </@Html.BeginForm()>
</body>
```

Annotations highlight the `ValidationSummary` call and the validation error message in the browser's developer tools.

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:

Födelsedatum:

Validering med *data annotation*

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

The diagram illustrates the validation process. On the left, a screenshot of a C# code editor shows a class definition with validation attributes:

```
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
        {
            get { return this.NextBirthdayDate.Year - this.Date.Year; }
            set { DaysUntilNextBirthday = value; }
        }
        int DaysUntilNextBirthday
        {
            get { return (Date - DateTime.Today).Days; }
            set { }
        }
    }
}
```

A blue arrow points from the [DataType(DataType.Date)] attribute to a callout box on the right. The callout box contains a heading and two validation error messages:

Nästa födelsedag

- * Namn måste anges.
- * Födelsedatum måste anges.

The validation errors are displayed above two input fields:

Namn:

Födelsedatum:

A note at the bottom left states: "Attributet DataType är inget valideringsattribut. Här ser det till att användargränssnittet undertrycker tiden och endast visar datumet."

Placering av felmeddelanden

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

```
@using (Html.BeginForm("NextBirthday", "Home", FormMethod.Post)) {  
    @Html.AntiForgeryToken()  
    @Html.ValidationSummary(true)  
    <div class="form-group">  
        @Html.LabelFor(model => model.Name)  
        @Html.EditorFor(model => model.Name)  
        @Html.ValidationMessageFor(model => model.Name)  
    </div>  
  
    <div class="form-group">  
        @Html.LabelFor(model => model.Birthdate)  
        @Html.EditorFor(model => model.Birthdate)  
        @Html.ValidationMessageFor(model => model.Birthdate)  
    </div>  
  
    <div class="form-group">  
        <input type="submit" value="Skicka" />  
    </div>  
}
```

Nästa födelsedag

Namn
Namn måste anges.

Födelsedatum
Födelsedatum måste anges.

Skicka

```
using System;  
using System.ComponentModel;  
using System.ComponentModel.DataAnnotations;  
using System;  
using System.ComponentModel;  
using System.ComponentModel.DataAnnotations;  
namespace NextBirthday.Models  
{  
    public class Birthday  
    {  
        [Required(ErrorMessage = "Födelsedatum  
[DataType(DataType.Date)]  
[Display Name = "Födelsedatum"]  
public DateTime Birthdate { get; set; }  
        [Required(ErrorMessage = "Namn")]  
[Display Name = "Namn måste ang  
public string Name { get; set; }  
        public int Age  
        {  
            get  
            {  
                return nextBirthday - Today  
            }  
        }  
    }  
}
```

Validering på klienten

- ✓ ASP.NET MVC 5 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.

```
    @using (Html.BeginForm())
    {
        @Html.AntiForgeryToken()
        @Html.ValidationSummary(true)
        <div class="form-group">
            @Html.LabelFor(model => model.Name)
            @Html.EditorFor(model => model.Name)
            @Html.ValidationMessageFor(model => model.Name)
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.Birthdate)
            @Html.EditorFor(model => model.Birthdate)
            @Html.ValidationMessageFor(model => model.Birthdate)
        </div>
        <div class="form-group">
            <input type="submit" value="Skicka" />
        </div>
    }
    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/jqueryval")
```

```
...iguration>
<appSettings>
    <add key="webpages:Version" value="5.0.0.0"/>
    <add key="ClientValidationEnabled" value="true"/>
    <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
</appSettings>
```

Namn	<input type="text" value="Ellen Nu"/>
Födelsedatum	<input type="text"/> Födelsedatum måste anges.
<input type="button" value="Skicka"/>	

Eget valideringsattribut

- ✓ Egna valideringsattribut för validering på servern skapar du genom att härleda en klass från ValidationAttribute och överskugga metoden IsValid.
- ✓ Attributet Required används för att bestämma om ett värde krävs eller inte varför IsValid ska returnera true när värdet är null.

```
public class BeforeTodayAttribute : ValidationAttribute
{
    public BeforeTodayAttribute()
        : base("The date must be later than todays date.")
    {
        // Empty!
    }

    public override bool IsValid(object value)
    {
        if (value == null)
        {
            return true;
        }

        return value is DateTime && ((DateTime)value) < DateTime.Today;
    }
}
```

```
public class Birthday
{
    [Required(ErrorMessage = "Födelsedatum måste anges.")]
    [BeforeToday(ErrorMessage = "Ett födelsedatum tidigare")]
    [DataType(DataType.Date)]
    [DisplayName("Födelsedatum")]
    public DateTime Birthdate { get; set; }

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

    Age
    ...
    this.NextBirthdayDate.Year
}
```

Validering på klienten genom valideringsattribut

- ✓ För att låta ett eget valideringsattribut även validera på klienten måste interfacet `IClientValidatable` implementeras och JavaScript för klientvalidering skrivas.

```
public class BeforeTodayAttribute : ValidationAttribute, IClientValidatable
{
    public BeforeTodayAttribute()
        : base("The date must be later than todays date.")
    {
        // Empty!
    }
    public override bool IsValid(object value)
    {
        if (value == null)
        {
            return true;
        }
        return value is DateTime & ((DateTime)value) < DateTime.Today;
    }
    public IEnumerable<ModelClientValidationRule> GetClientValidationRules(ModelMetadata metadata, ControllerContext context)
    {
        yield return new ModelClientValidationRule
        {
            ErrorMessage = FormatErrorMessage(metadata.GetDisplayName()),
            ValidationType = "beforetoday"
        };
    }
}
```

```
(function ($) {
    $.validator.addMethod("isDateLaterThanToday", function (value, element, params) {
        if (!/Invalid|NaN/.test(new Date(value))) {
            return new Date(value) < new Date();
        }
        return false;
    }, '');

    $.validator.unobtrusive.adapters.add("beforetoday", {}, function (options) {
        options.rules['isDateLaterThanToday'] = true;
        options.messages['isDateLaterThanToday'] = options.message;
    });
})(jQuery);
```

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

Mer information

- ✓ I kurslitteraturen kapitel 22-25 hittar du mer information om "*model templates*", bindning, validering och "*unobtrusive Ajax*".
- ✓ På <http://www.asp.net/mvc/overview/getting-started/introduction/adding-validation> finns exempel på validering. På <http://www.asp.net/tutorials/older-versions/models-data/performing-simple-validation-cs> finns flera "gamla" exempel på hur validering kan implementeras.