

# Unobtrusive Ajax



# 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 Look, förutom Linnéuniversitetets logotyp och symbol, 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 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.

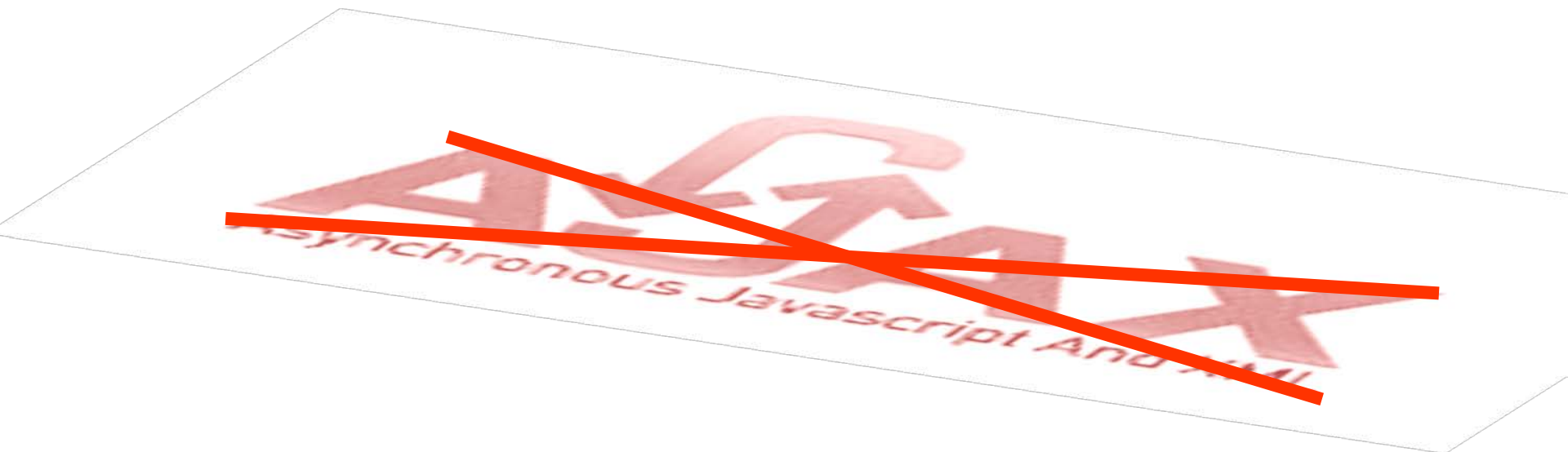
# Använd Ajax när du...

- ✓ ...har komplexa sidor och bara behöver uppdatera en liten del av sidan.
- ✓ ...vill att användare ska uppfatta att din applikation har korta svarstider.
- ✓ ...vill reducera belastningen av webbservern och databasserver.
- ✓ ...har behov av asynkrona anrop.



# Använd INTE Ajax när...

- ✓ ...din applikation blir beroende av att användare tillåter JavaScript. Användare ska kunna använda applikationen även om de inte tillåter JavaScript.
- ✓ ...det negativt påverkar användarens upplevelse när historikknappar och bokmärken används i webbläsaren.
- ✓ ...det inte finns goda skäl för det. Det tar tid att ladda JavaScript, och det tar tid att exekvera dem.



# ASP.NET MVC och ”*Ajax Helpers*”

- ✓ MVC har stöd för Ajax-anrop. Från och med version 3 finns även stöd för ”unobtrusive Ajax” baserad på jQuery.
- ✓ För att ” unobtrusive Ajax” ska fungera på klienten måste...
  - ...UnobtrusiveJavaScriptEnabled vara satt till true, vilken den är som standard i Web.config.
  - ...jquery-1.8.1.min.js , eller annan lämplig version, länkas in.
  - ...jquery.unobtrusive-ajax.min.js länkas in.
- ✓ Med hjälp av ”*Ajax Helpers*” är det enkelt att skapa förutsättningar för asynkrona förfrågningar.
  - Ajax.ActionLink
  - Ajax.BeginForm
  - Ajax.RouteLink
  - Ajax.BeginRouteForm

# Ett traditionellt formulär

- ✓ Används `Html.BeginForm` renderas ett traditionellt formulär som postas tillbaka till servern och sidan renderas om fullständigt.

```
@model System.String
@{
    ViewBag.Title = "Hej!";
}
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    @Html.TextBox("greeting")
    <input type="submit" value="Posta hälsning (ingen Ajax-förfrågan)" />
}
<div id="result">
    @Model
</div>
```

```
<form action="/" method="post">
    <input name="__RequestVerificationToken" type="hidden" value="..." />
    <input id="greeting" name="greeting" type="text" value="" />
    <input type="submit" value="Posta hälsning (ingen Ajax-förfrågan)" />
</form>
<div id="result">
</div>
```

# Formulär som använder Ajax

- ✓ Genom att använda `Ajax.BeginForm` renderas ett formulär som använder Ajax och endast innehållet i specificerat element ersätts. I övrigt är allt precis som då ett traditionellt formulär renderas.

```
@model System.String

@{
    ViewBag.Title = "Hej!";
    var ajaxOptions = new AjaxOptions
    {
        UpdateTargetId = "result",
        Url = Url.Action("HelloAjax")
    };
}

@using ((Ajax.BeginForm("HelloAjax", ajaxOptions))
{
    @Html.AntiForgeryToken()
    @Html.TextBox("greeting")
    <input type="submit" value="Posta hälsning (med Ajax-förfrågan?" />
}
<div id="result">
    @Html.Partial("_Greeting")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

```
<input type="text" name="greeting" value="" data-ajax-mode="replace" data-ajax-update="#result" data-ajax-url="/Hello/HelloAjax" />
<input type="submit" value="Posta hälsning (med Ajax-förfrågan?" />
</form>
<div id="result">
    @Html.Partial("_Greeting")
</div>
<script src="/Scripts/jquery.unobtrusive-ajax.js"></script>
<script src="/Scripts/jquery.validate.js"></script>
<script src="/Scripts/jquery.validate.unobtrusive.js"></script>
```

# Elementär hantering av Ajax-förfrågan

- ✓ Precis som en vanlig förfrågan hanteras en Ajax-förfrågan av en publik metod i en controllerklass. I sin enklaste form returnerar en sådan metod en sträng som ersätter innehållet i specificerat element.



```
[HttpPost]
[ValidateAntiForgeryToken]
public string HelloAjax(string greeting)
{
    return "Din Ajax-hälsning: " + greeting;
}
```

```
<form action="/" data-ajax="true" data-ajax-method="post"
  id="form0" method="post">
  <input name="__RequestVerificationToken" type="hidden" value="..." />
  <input id="greeting" name="greeting" type="text" value="" />
  <input type="submit" value="Posta hälsning (med Ajax-förfrågan)" />
</form>
<div id="result">
  Din Ajax-hälsning: Tjabba, tjena, hallå!
</div>
```

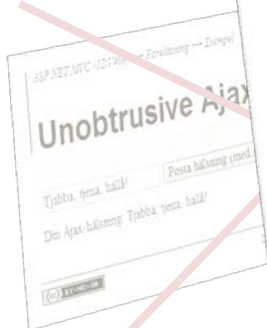


# Men vad händer om JavaScript är avstängt?

- ✓ Om JavaScript är avstängt...
  - ...anropas aldrig `onsubmit`, varför...
  - ...formuläret istället använder det normala beteendet, d.v.s. det postar sig själv tillbaka till servern, och...
  - ...allt fungerar utan att några fel inträffar, men webbläsaren visar endast det som controllermetoden returnerade.

**Din Ajax-hälsning: Tjabba, tjena, hallå!**

```
[HttpPost]
[ValidateAntiForgeryToken]
public string HelloAjax(string greeting)
{
    return "Din Ajax-hälsning: " + greeting;
}
```



# En controllermetod – olika typer av förfrågningar

- ✓ Controllermetoden måste kunna hantera både en Ajax-förfrågan och en vanlig förfrågan. Genom att använda metoden `Request.IsAjaxRequest()` kan du ta reda på vilket sätt förfrågan görs.
  - Är det fråga om en Ajax-förfrågan returneras en sträng med hjälp av `Content`.
  - Är det en "vanlig" förfrågan returneras en vy som vanligt. Observera att det inte går att skicka med modellen på vanligt sätt i detta fall på grund av den modellens typ är av typen `String`, vilket medför att modelldata tolkas som en "*page layout*". Egenskapen `ViewData.Model` får istället användas.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult HelloAjax(string greeting)
{
    if (Request.IsAjaxRequest())
    {
        return Content("Din Ajax-hälsning: " + greeting);
    }

    ViewData.Model = greeting;
    return View("Index");
}
```

# Ajax och "partial views"

- ✓ Istället för att "bara" returnera en sträng kan en partiell vy returneras, som även används då det inte är en Ajax-förfrågan.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Index(string greeting)
{
    if (Request.IsAjaxRequest())
    {
        return PartialView("_Greeting", greeting);
    }

    ViewData.Model = greeting;
    return View("Index");
}
```

```
@model System.String

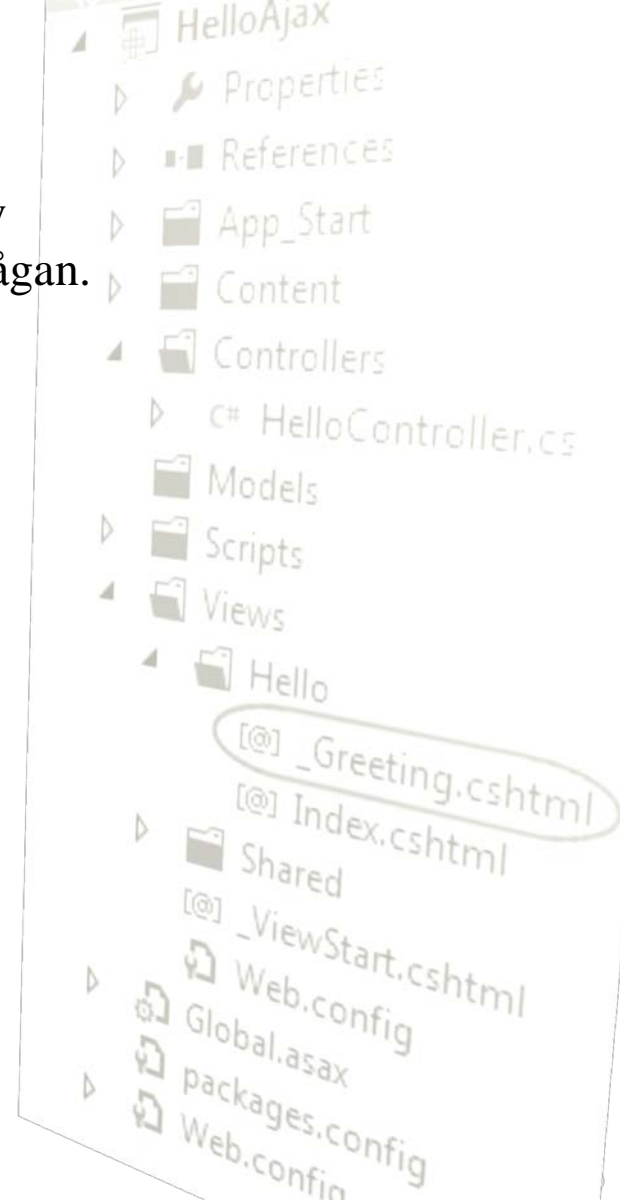
<p>
    Din hälsning: <strong>@Model</strong>
</p>
```

```
@{
    ViewBag.Title = "Hej!";
    var ajaxOptions = new AjaxOptions
    {
        UpdateTargetId = "result",
        Url = Url.Action("Index")
    };
}

@using (Ajax.BeginForm(ajaxOptions))
{
    @Html.AntiForgeryToken()
    @Html.TextBox("greeting")
    <input type="submit" value="Posta hälsning" />
}

<div id="result">
    @Html.Partial("_Greeting")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```



# Enkel hantering av fel vid Ajax-förfrågan

- ✓ Det är viktigt att `Response.StatusCode` sätts till lämplig felkod så att JavaScript-funktionen som definieras av `OnFailure` körs på klienten.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Index(GreetData greetData)
{
    if (greetData.Greeting.Length % 2 == 0)
    {
        ModelState.AddModelError("Greeting",
            "Strängen kan inte innehålla ett jämt antal tecken.");
    }

    if (Request.IsAjaxRequest())
    {
        if (ModelState.IsValid)
        {
            return PartialView("_Greeting", greetData.Greeting);
        }
        else
        {
            Response.StatusCode = 400;
            return Content("Ett fel inträffade.");
        }
    }

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

```
@model HelloAjax.Models.GreetData

@{
    ViewBag.Title = "Hej!";
    var ajaxOptions = new AjaxOptions
    {
        UpdateTargetId = "result",
        Url = Url.Action("Index"),
        OnFailure = "handleAjaxFailure"
    };
}

@using (Ajax.BeginForm(ajaxOptions))
{
    @Html.ValidationSummary(true);
    @Html.AntiForgeryToken()
    @Html.TextBoxFor(m => m.Greeting)
    @Html.ValidationMessageFor(m => m.Greeting)
    <input type="submit" value="Posta hälsning" />
}

<div id="result">
    @if (Model != null)
    {
        @Html.Partial("_Greeting", Model.Greeting)
    }
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")

    <script>
        function handleAjaxFailure(ajaxContext) {
            // Replace the contents of #result element with the response text (error message).
            $("#result").html("<span class='field-validation-error'>" + ajaxContext.responseText + "</span>");
        }
    </script>
}
```

# Hantering av fel vid Ajax-förfrågan med JSON

- ✓ För att skicka samtliga felmeddelanden, gällande så väl modellens egenskaper som formuläret i sig, till klienten kan JSON användas.
- ✓ På klienten används `jQuery.validate().showErrors()` för att visa felen där de ska visas.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Index(GreetData greetData)
{
    if (greetData.Greeting.Length % 2 == 0)
    {
        ModelState.AddModelError("Greeting",
            "Strängen kan inte innehålla ett jämt antal tecken.");
    }

    if (Request.IsAjaxRequest())
    {
        if (ModelState.IsValid)
        {
            return PartialView("_Greeting", greetData.Greeting);
        }

        Response.StatusCode = 400;

        // Transform the modelstate errors to a dictionary where property names is
        // associated with there errors.
        var errors = ModelState
            .Where(kvp => kvp.Value.Errors.Any())
            .ToDictionary(kvp => kvp.Key,
                kvp => kvp.Value.Errors.Select(e => e.ErrorMessage).ToArray());

        return Json(new { Errors = errors }, JsonRequestBehavior.AllowGet);
    }

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

```
@model HelloAjax.Models.GreetData

@{
    ViewBag.Title = "Hej!";
    var ajaxOptions = new AjaxOptions
    {
        UpdateTargetId = "result",
        Url = Url.Action("Index"),
        OnFailure = "handleAjaxFailure"
    };
}

@using (Ajax.BeginForm(ajaxOptions))
{
    @Html.ValidationSummary(true);
    @Html.AntiForgeryToken()
    @Html.TextBoxFor(m => m.Greeting)
    @Html.ValidationMessageFor(m => m.Greeting)
    <input type="submit" value="Posta hälsning" />

    <div id="result">
        @if (Model != null)
        {
            @Html.Partial("_Greeting", Model.Greeting)
        }
    </div>

    @section Scripts {
        @Scripts.Render("~/bundles/jqueryval")

        <script>
            function handleAjaxFailure(ajaxContext) {
                // Empty the ajax target element.
                $('#result').empty();

                // Convert the response text, a JSON string, into a dictionary.
                var errors = JSON.parse(ajaxContext.responseText)["Errors"];

                // Show property errors.
                $('form').validate().showErrors(errors);
            }
        </script>
    }
}
```



# ...används ValidationSummary så blir det komplexare

- ✓ Används ValidationSummary för att visa fel som inte är knutna till någon egenskap i modellen, medför det att skriptet som visar fel blir mer omfattande.

```
@model HelloAjax.Models.GreetData
@{
    ViewBag.Title = "Hej!";
    var ajaxOptions = new AjaxOptions
    {
        UpdateTargetId = "result",
        Url = Url.Action("Index"),
        OnSuccess = "handleAjaxSuccess",
        OnFailure = "handleAjaxFailure"
    };
}

<script>
    function handleAjaxSuccess(ajaxContext) {
        // Remove all none property errors.
        if ($('#div[data-valmsg-summary="true"]').length != 0) {
            if ($('#div[data-valmsg-summary="true"]').hasClass("validation-summary-errors")) {
                $('#div[data-valmsg-summary="true"]').removeClass("validation-summary-errors");
                $('#div[data-valmsg-summary="true"]').addClass("validation-summary-valid");
            }
            $('#div[data-valmsg-summary="true"] ul').empty();
        }
    }

    function handleAjaxFailure(ajaxContext) {
        // Empty the ajax target element.
        $('#result').empty();

        // Convert the response text, a JSON string, into a dictionary.
        var errors = JSON.parse(ajaxContext.responseText)["Errors"];

        // Show property errors.
        $('#form').validate().showErrors(errors);

        // Show none property errors. Insert validation summary elements, if we need to, and append
        // form errors to the validation summary.
        if (errors[""] != undefined) {
            if ($('#div[data-valmsg-summary="true"]').length == 0) {
                $('#form').prepend('<div class="validation-summary-errors" data-valmsg-summary="true"><ul></ul></div>');
            }
            else if ($('#div[data-valmsg-summary="true"]').hasClass("validation-summary-valid")) {
                $('#div[data-valmsg-summary="true"]').removeClass("validation-summary-valid");
                $('#div[data-valmsg-summary="true"]').addClass("validation-summary-errors");
            }
            $('#div[data-valmsg-summary="true"] ul').empty();
            for (var i in errors[""]) {
                $('#div[data-valmsg-summary="true"] ul').append("<li>" + errors[""][i] + "</li>");
            }
        }
    }
}
</script>
```