

# Webbformulär

# 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:

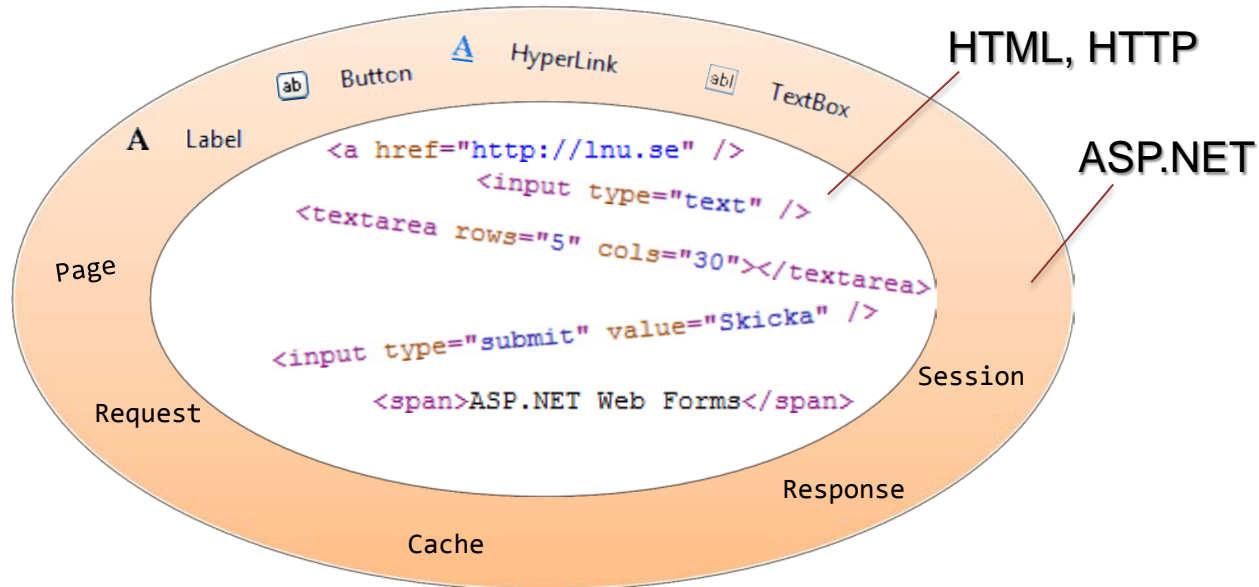
- kopiera hela eller delar av innehållet
- sprida hela eller delar av innehållet
- visa hela eller delar av innehållet offentligt och digitalt
- konvertera innehållet till annat format
- du får även göra om innehållet

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

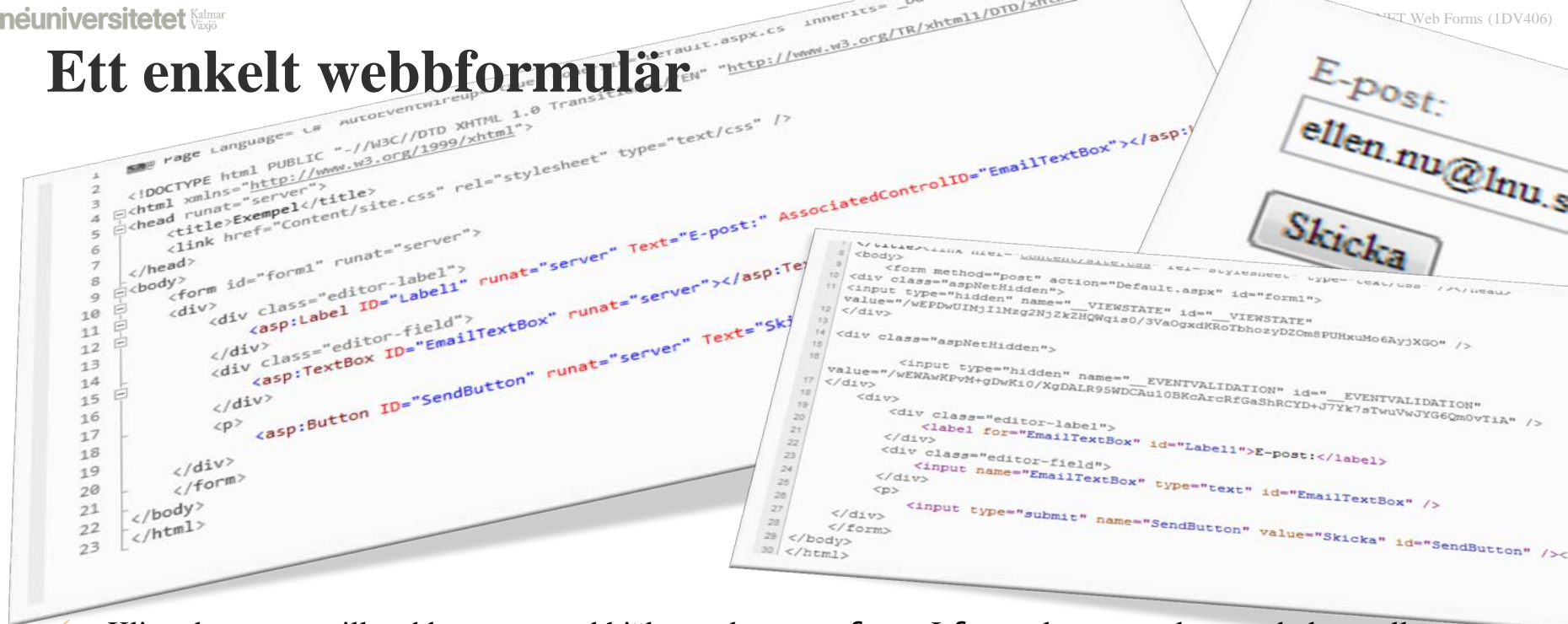
Vid all användning måste du ange källan: "Linnéuniversitetet – ASP.NET MVC" och en länk till <https://coursepress.lnu.se/kurs/aspnet-mvc> och till Creative Common-licensen här ovan.

# Vad gör ASP.NET?

- ✓ ASP.NET lägger ett abstraktionslager ovanpå HTML och HTTP i form av olika typer, som t.ex. `Page`, `HttpRequest` och `HttpResponse`.
- ✓ Tack vare abstraktionslagret kan du som programmerare därför arbeta med objekt och händelser på webbservern.
  - (Detta har visat sig vara ett bra(?) och beprövat koncept som t.ex. programmerare av Windowsapplikationer använt länge.)



# Ett enkelt webbformulär



- ✓ Klientdata postas till webbservern med hjälp av elementet form. I form-elementet placerar du kontroller som representerar användargränssnittet, som kommandoknappar, alternativknappar, kryssrutor och textfält.
- ✓ Då användaren fyllt i textfältet och postat formuläret, d.v.s. gjort en s.k. "postback", kan innehållet användaren matade in i textfältet hämtas på olika sätt på servern:

~~`string emailAddress = Request.Form["EmailTextBox"];`~~  
 Inte ett speciellt fullständigt objektorienterat sätt att hämta data från formuläret.

`string emailAddress = EmailTextBox.Text;`  
 Ett mycket bättre och intuitivare sätt! Då ett formulär postas tillbaka till ASP.NET extraheras värdena och kontrollobjekten skapas och initieras.

# Element på klienten - objekt på servern



klient

E-post:

```
<input name="EmailTextBox" type="text" id="EmailTextBox" />
```

`abl` **TextBox**

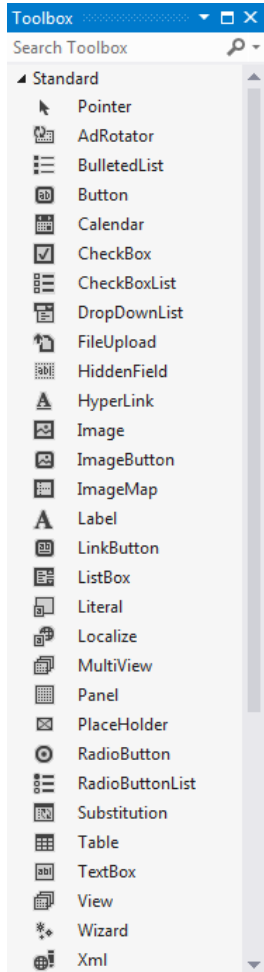
```
<asp:TextBox ID="EmailTextBox" runat="server">
```



webbserver

- ✓ Klienten arbetar med HTML, t.ex. ett textfält är fortfarande inget annat än ett `input`-element där `type` är satt till `text`.
- ✓ På webbservern representeras textfältet av ett objekt av typen `TextBox`, som är en serverkontroll. Objekt av typen `TextBox` renderas till klienten som ett `input`-element.

# Fördelar med kontroller



- ✓ I "code behind"-filer är det enkelt att arbeta med kontrollerna, som är objekt instansierade från klasser, som t.ex. `TextBox`.
- ✓ Kontroller "gömmar" HTML. Ett `TextBox`-objekt renderas som ett `input`-element.
- ✓ Den HTML som kontrollerna renderar anpassas automatiskt efter klientens webbläsare.
- ✓ Ett `Panel`-objekt kan beroende av klient renderas som en `div`-element eller en `table`-element.

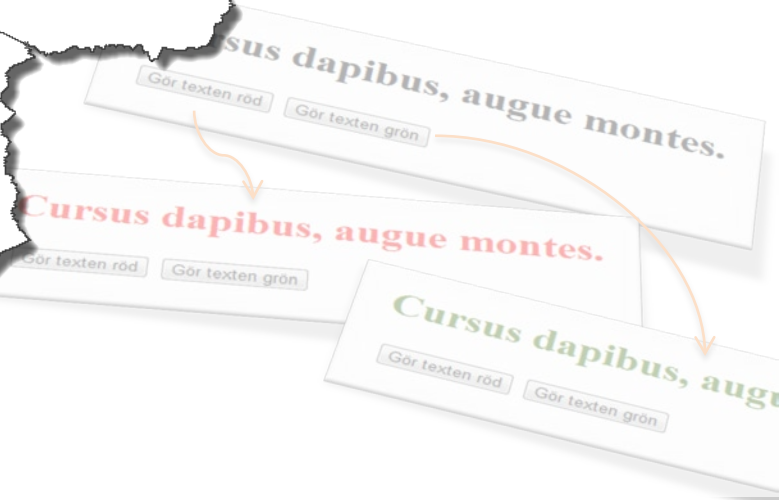
# Serverkontroller gör det enklare för dig

- ✓ Du kan enkelt hämta (och sätta) textinnehållet i en TextBox genom att använda egenskapen Text. Men du kan göra så mycket mer med alla de egenskaper de olika serverkontrollerna har...
- ✓ En Label-kontrolls egenskap CssClass (renderas till klienten som attributet class) kan t.ex. enkelt ändras med kod i "code behind"-filen.

```
10 </form id="form1" runat="server">
11 <div>
12 <h1>
13 <asp:Label ID="MyLabel" runat="server">Cursus dapibus, augue montes.</asp:Label></h1>
14 <asp:Button ID="RedButton" runat="server" Text="Gör texten röd" OnClick="RedButton_Click" />
15 <asp:Button ID="GreenButton" runat="server" Text="Gör texten grön" OnClick="GreenButton_Click" />
16 </div>
17 </form>
18 </body>
19 </html>

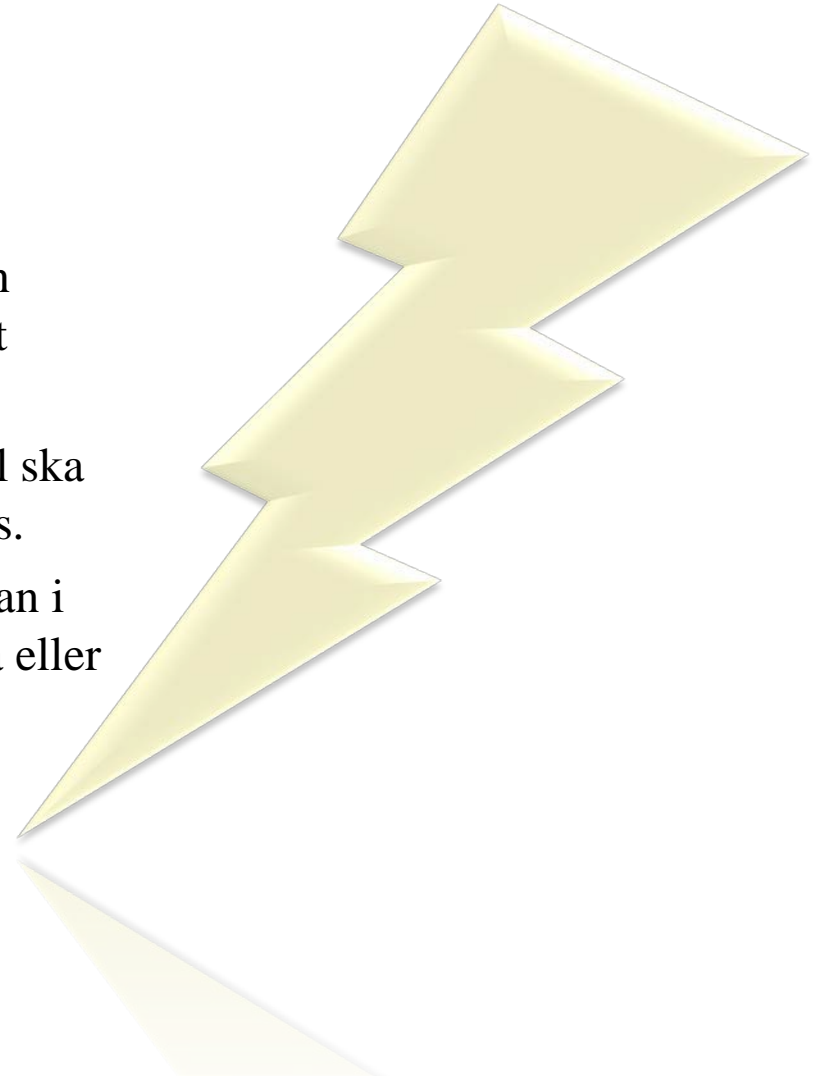
14
15 protected void RedButton_Click(object sender, EventArgs e)
16 {
17     MyLabel.CssClass = "red";
18 }
19
20 protected void GreenButton_Click(object sender, EventArgs e)
21 {
22     MyLabel.CssClass = "green";
23 }
24 }
```

```
1 .green
2 {
3     color: #336600;
4 }
5 .red
6 {
7     color: #FF0000;
8 }
```



# Händelsestyrd programmering

- ✓ ASP.NET använder en händelsestyrd programmeringsmodell, "event-driven programming model".
- ✓ ASP.NET skapar händelser ("event") du kan abonnera på genom hanterarmetoder ("event handlers").
- ✓ I hanterarmetoden skriver du den kod du vill ska exekveras då en specifik händelse har utlösts.
- ✓ Ett stort antal händelser utlöses på serversidan i samband att en klient gör en GET av en sida eller en "postback" av formuläret tillbaka till webbservern.





# Första efterfrågan



klient



webbserver

1. Sidan efterfrågas för första gången. ASP.NET skapar sidan och kontrollerna. Eventuella initiala värden sätts. Sidan renderas ut i HTML till klienten. Objekten förstörs och minnet lämnas tillbaka till servern.
2. Användaren gör något som leder till en "postback", kanske trycker på en knapp. Formuläret (sidan) med allt data skickas tillbaka till webbservern.
3. ASP.NET (åter)skapar de objekt som är kopplade till sidan, återställer kontrollernas värden som det var då sidan skickades ut till klienten senast.
4. ASP.NET undersöker vad som orsakade att en "postback" skedde, och utlöser lämpliga händelser. Typiskt tar du hand om händelserna och uppdaterar kanske objekten med nya värden.
5. Den modifierade sidan renderas till HTML och skickas till klienten. Alla objekt förstörs och samtliga resurser som behandlingen av sidan krävde lämnas tillbaka.

# View State

Exempel

```

7 </title><link href="Content/site.css" rel="stylesheet" type="text/css" /></head>
8 <body>
9     <form method="post" action="Default.aspx" id="form1">
10     <div class="aspNetHidden">
11     <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
12     value="/wEPDwUIMjI1Mzg2NjZkZHQWqis0/3VaOgxdKRoTbhozyDZOm8PUHxuMo6AyjXGO" />
13     </div>
14     <div class="aspNetHidden">
15

```

- ✓ För kunna implementera händelsestyrd programmeringsmodell måste ASP.NET kunna "komma ihåg" en sidas status – det är precis det som "view state" har till uppgift.
- ✓ ASP.NET undersöker alla kontrollers egenskaper och serialiserar dessa till en Base64-sträng, som därefter lagras i ett dolt `input`-element.
- ✓ Då sidan senare skickas tillbaka till webbservern kan ASP.NET genom att jämföra formulärets data med det data som serialiserats till "view state" utlösa händelser.
- ✓ Genom att använda "view state" kan resurser frigöras på webbservern, som då inte behöver lagra information om alla sidors olika status.

# Vad händer då formuläret postas?

1. Då en "postback" sker skapar ASP.NET sidan och kontrollobjekten och ger kontrollerna de standardvärden som finns definierade i aspx-sidan. Just nu är sidan likadan som användaren först såg den första gången.
2. ASP.NET läser informationen i "view state" och uppdaterar kontrollerna. Nu är sidan som den senast såg ut för användaren.
3. ASP.NET kontrollerar nu om användaren har ändrat något i formuläret och tilldelar kontrollerna de nya värdena. Nu är sidan och dess objekt i fas med vad användaren såg när han/hon gjorde sin "postback".
4. Eventuella hanterarmetoder du skrivit exekveras.
5. ASP.NET skapar en ny "view state"-sträng, renderar sidan och skickar tillbaka den till klienten.

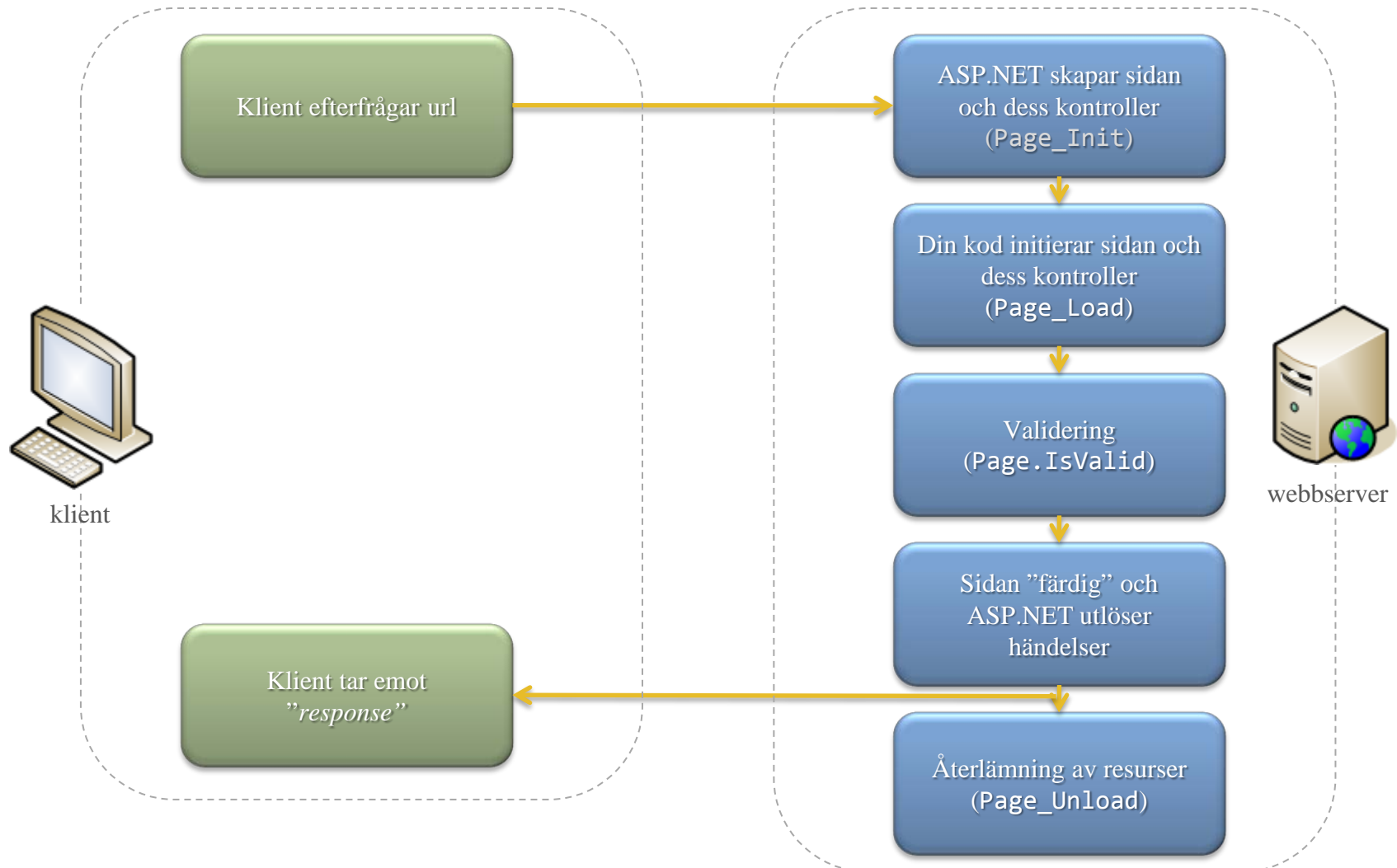


webbserver



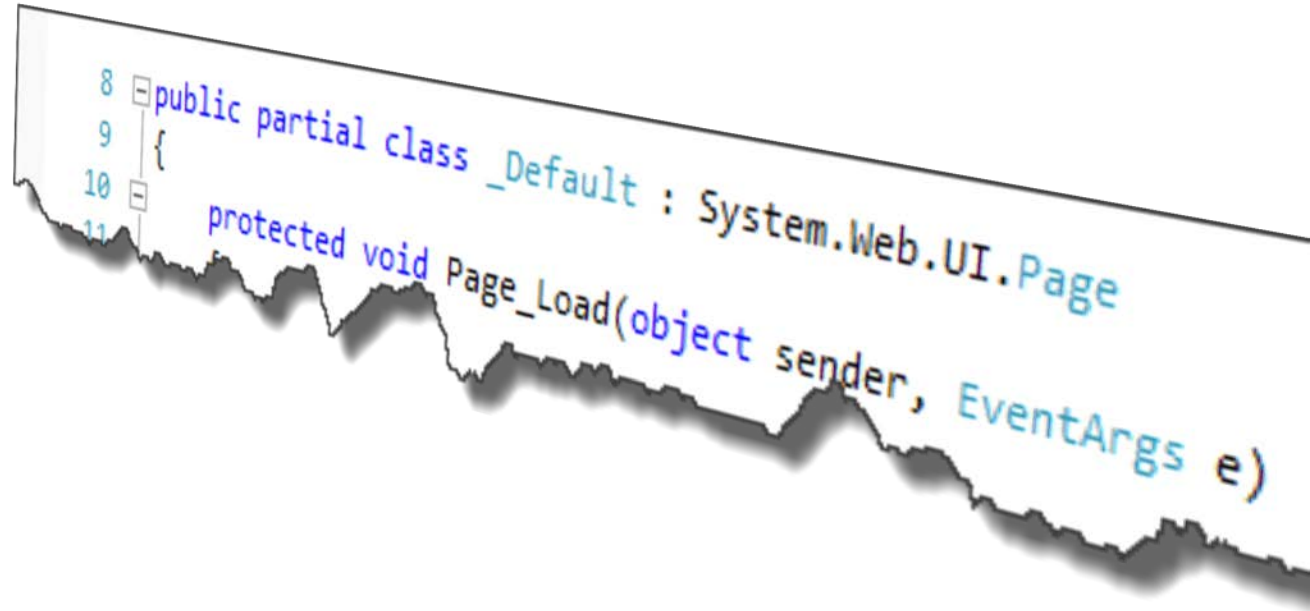
klient

# Ett webbformulärs liv och leverne



# Klassen Page

- ✓ Alla webbformulär är **Page**-objekt.
- ✓ Klassen i en "code-behind"-fil ärver från klassen **Page**, som har mycket funktionalitet redan från början.
- ✓ Genom att ärva från **Page** får du bl.a. tillgång till följande egenskaper:
  - Session
  - (Application)
  - Cache
  - Request
  - Response
  - Server
  - User
  - Trace



```

8 public partial class _Default : System.Web.UI.Page
9 {
10
11     protected void Page_Load(object sender, EventArgs e)
    
```

# Session

- ✓ Sparar användarspecifik data på servern mellan "requests".
  - användarinformation
  - kundvagn
- ✓ Egenskapen `Session` är av typen `SessionHttpState`.
- ✓ Datat sparas i en associativ array ("dictionary").

```
// Spara referens till objekt i en sessionsvariabel.
Session["shoppingCart"] = myShoppingCart;

// Tilldela referensvariabel referens från sessionsvariabel.
myShoppingCart = Session["shoppingCart"] as ShoppingCart;
```

# Application

- ✓ Fungerar precis som **Session** men är global för hela applikationen inte bara knuten till en användare och session.
- ✓ Använd **ALDRIG Application**.  
Använd istället **Cache**, som är mycket mer kompetent.



# Cache

- ✓ Cache är till för att hantera data som är global för hela applikationen. Är mer ”intelligent” än **Application**.
  - Du kan t.ex. bestämma hur länge datat ska ”cachas”.
  - Om minnet håller på att ta slut tas datat automatiskt bort från ”cachen”. Genom att sätta prioritet på datat får du inflytande över vilket data som ska tas bort.
- ✓ Är en associativ array (”dictionary”) och fungerar syntaxmässigt på samma sätt som **Session**.





# Request

- ✓ Egenskapen **Request** innehåller information om den "HTTP request" som orsakade sidan att laddas, bl.a. finns här information som skickades av klienten.
- ✓ Datat i **Request**-objektet exponeras av ASP.NET med hjälp av flera egenskaper.
- ✓ (Användes betydligt flitigare i klassisk ASP.)



# Exempel på egenskaper i klassen HttpRequest

- ✓ **ApplicationPath / PhysicalPath**
  - Pekar på en applikations virtuella sökväg / absoluta sökväg.
- ✓ **Browser**
  - Talar om t.ex. vilket stöd en webbläsare har (cookies, ActiveX m.m.).
- ✓ **Cookies**
  - Vilka cookies finns?
- ✓ **QueryString**
  - De värden som kan komma via en ”query string”.
- ✓ **UserAgent**
  - Vilken webbläsare använder klienten.
- ✓ **UserHostAddress / UserHostName**
  - IP-adress och DNS namn.

# Response

- ✓ Representerar datat webbservern kommer att skicka till klienten som gjort en förfrågan.
- ✓ Är av typen `HttpResponse`.
- ✓ I klassisk ASP var `Response`-objektet enda sättet att skriva ut text till klienten (`Response.Write` – funkar såklart i ASP.NET också).



# Exempel på medlemmar i klassen `HttpResponse`

- ✓ `Write()`
  - Skriver ut till strömmen
- ✓ `Redirect()`
  - Skickar användaren till en annan sida genom en "round trip".
- ✓ `Cookies`
  - Samling för de "cookies" som ska skickas med till klienten.

# Server – några metoder

- ✓ Är av typen `System.Web.HttpServerUtility`.
- ✓ `GetLastError()`
  - Återger senast kastade undantag. Används i så kallade ”Application Events”.
- ✓ `HtmlEncode()/HtmlDecode()`
  - Mycket användbart när ett formulärs data hanteras. Används t.ex. för att ta hand om HTML-taggar för att kunna behandla dessa som strängar.
- ✓ `MapPath()`
  - Returnerar fysiska sökvägen till aspx-filen på servern.
- ✓ `Transfer()`
  - Skickar iväg användaren till en annan sida utan ”round trip”.

# User och Trace

- ✓ User-objektet, som representerar en autentiserad användare, kommer du inte i kontakt med under denna kurs utan först då du väljer att börja titta på säkerhet och är i behov av att upprätta olika regler för olika användare.
- ✓ Trace används dels för att hitta felaktigheter men även för att studera prestanda och tillgänglighet i sin applikation. Du kan läsa om Trace i kurslitteraturen på sidorna 121 – 126.

The screenshot shows the ASP.NET Trace tool interface. It displays the following sections:

- Request Details:**
  - Session Id: pddbvXSpaZush3wpigf1v0fv
  - Time of Request: 2011-03-29 11:09:08
  - Request Encoding: Unicode (UTF-8)
  - Request Type: GET
  - Status Code: 200
  - Response Encoding: Unicode (UTF-8)
- Trace Information:**

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit	1,38963500505205E-05	0,000014
aspx.page	End PreInit	2,63299264115126E-05	0,000012
aspx.page	Begin Init	3,94948896172689E-05	0,000013
aspx.page	End Init	4,82715317544398E-05	0,000009
aspx.page	Begin InitComplete	5,70481738916107E-05	0,000009
aspx.page	End InitComplete	6,80189765630742E-05	0,000011
aspx.page	Begin PreLoad	7,67956187002451E-05	0,000009
aspx.page	End PreLoad	8,5572260837416E-05	0,000009
aspx.page	Begin Load	0,000126529924144213	0,000009
aspx.page	End Load	0,000148837222909523	0,000014
aspx.page	Begin LoadComplete	0,000157613865046694	0,000009
aspx.page	End LoadComplete	0,000170413134830068	0,000009
aspx.page	Begin PreRender	0,000179189776967239	0,000009
aspx.page	End PreRender	0,000187966419104409	0,000009
aspx.page	Begin PreRenderComplete	0,000357282473667331	0,000009
aspx.page	End PreRenderComplete	0,000528426995342163	0,000169
aspx.page	Begin SaveState	0,000543054732237448	0,000171
aspx.page	End SaveState	0,000552197067797001	0,000015
aspx.page	Begin SaveStateComplete	0,000560608016511789	0,000009
aspx.page	End SaveStateComplete	0,000823907280626916	0,000008
aspx.page	Begin Render		0,000263
aspx.page	End Render		
- Control Tree:**

Control UniqueID	Type	Render Size Bytes (including children)	ViewState Size Bytes (excluding children)	ControlState Size Bytes (excluding children)
_Page	ASP.default_aspx	1122	0	0
ctl03	System.Web.UI.LiteralControl	172	0	0
ctl00	System.Web.UI.HtmlControls.HtmlHead	105	0	0