Microsoft

Bygg en webbplats NU!

Microsoft®

Visual Web Developer™ 2005 Express Edition

Komplett med programvara och konkreta exempel!

Jim Buyens

Swedish Language Edition published by Docendo Sverige AB

Kapitel 2

Lär känna programmeringsmodellen bakom ASP.NET 2.0

ASP.NET utgör en stor del av Microsofts initiativ till .NET. I jämförelse med tidigare tekniker som till exempel Microsoft® Active Server Pages (ASP) och PHP Hypertext Processor (PHP), är ASP.NET mycket bredare, mycket mer kraftfull och tillförlitlig. I och med dessa förbättringar inom ASP.NET betraktar Microsoft tekniken med Active Server Pages som en teknik som så småningom kommer att försvinna.

I jämförelse med tidigare versioner tillhandahåller ASP.NET 2.0 många förbättringar vad gäller såväl styrka som användarvänlighet. Samtliga versioner av ASP.NET levererar webbsidor på ungefär samma sätt. I nästa avsnitt förklaras det hur detta fungerar. I ett senare avsnitt förklaras grundtyperna i programvarukomponenterna du lägger till i en webbsida inom ASP.NET.

Men glöm inte bort att detta kapitel endast ger en enklare introduktion till dessa ämnen. Det finns böcker som skrivits endast om dessa och materialet i denna bok ska bara hjälpa dig att komma igång med Microsoft Visual Web Developer™.

Om du tycker att denna introduktion verkar vara lite abstrakt eller svår, hoppar du över den och går direkt till kapitel 3. Du kan alltid gå tillbaka till detta kapitel senare.

Följa en webbsida genom ASP.NET 2.0

I det stora hela körs ASP.NET på en webbserver. För varje gång en besökare anropar en sida som har ett filtnamnsillägg som till exempel .aspx, kommer ASP.NET göra följande:

1. Ladda den begärda sidan till webbserverns minne.
2. Exekvera eventuella mjukvarukomponenter från ASP.NET som webbsidan innehåller.
3. Skicka resultatsidan till besökaren.

När en webbsida laddas in i minnet skiljer ASP.NET på två väsentligt skilda typer av innehåll.

- **Vanliga HTML-taggar** är just vad de är: vanlig HTML-kod. ASP.NET gör ingenting med dessa, förutom att lagra dem och senare skicka dem precis som de ser ut. Om din sida till exempel innehåller följande tagg:

```
<img src="images/mylogo.gif">
```

  kommer ASP.NET att skicka:

```
<img src="images/mylogo.gif">
```

  till besökaren. Webbservern ändrar inte vanlig HTML-kod över huvud taget.

- **Serverkontroller för ASP.NET** är mjukvarukomponenter som ASP.NET laddar in i minnet. Följande tagg instruerar till exempel ASP.NET att ladda serverkontrollen *HtmlImage* till minnet.

```
<img src="images/mylogo.gif" id="imgMyLogo" runat="server" />
```

  Den viktigaste skillnaden mellan denna kod och den tidigare, är attributet *runat="server"*, som instruerar ASP.NET att skapa en serverkontroll på webbservern.

- Taggnamnet (i detta fall *img*) instruerar ASP.NET vilken typ av serverkontroll som ska skapas.
- Attributet *id*= tillhandahåller ett namn till vilket andra serverkontroller på samma sida kan referera till.

- ASP.NET skickar attributet *src=* (och alla andra som är närvarande) till serverkontrollen som input.
- Slutttecknet ”/” är likställt med slutkod i XML.

Så snart ASP.NET slutar att ladda serverkontrollen, överger den XML:en som skapade den. När det är dags att skicka sidan till besökaren, skickar ASP.NET *inte* serverkontrollens XML-kod i original. I stället anropas en funktion (eller rättare sagt en metod – *method*) med namnet *Render,* som är en del av serverkontrollen, och därefter sänds resultatet av den metoden till besökaren.

Även om samtliga serverkontroller som levereras med ASP.NET 2.0 renderar sin output som HTML, lägger inte ASP.NET några restriktioner på resultatet från metoden *Render* i en serverkontroll. Resultatet behöver inte ens likna originaltaggen i XML-format, och behöver inte ens vara i HTML-format. Det kan till exempel bestå av en bildfil eller ett Microsoft Word-dokument.

NOTERA

Alla serverkontroller inom ASP.NET har en *Render*-metod.

För att summera livscykeln för en sida som skapas med ASP.NET:

1. En besökare begär en fil som har filnamnstillägget .aspx.
2. ASP.NET läser filen från webbserverns filsystem.
3. ASP.NET inspekterar varje tagg i filen och laddar detta till minnet.
   - Om taggen innehåller attributet *runat=”server”* laddar ASP.NET en mjukvarukomponent som kallas serverkontroll. Namnet på taggen avgör vilken typ av serverkontroll som avses.
   - Taggar utan attributet *runat=”server”* hanteras som vanlig HTML. ASP.NET laddar dessa till en mjukvarukomponent som vid rätt tillfälle skickar originaltaggen till besökarens webbläsare.
4. När alla taggar laddats in i minnet kör ASP.NET den angivna programkoden inom respektive serverkontroll. Denna v kan komma åt resurser på webbservern; det kan förändra innehållet eller egenskaperna på alla serverkontroller på sidan och det kan till och med lägga till eller ta bort andra serverkontroller.

TIPS

ASP.NET hanterar alla serverkontrolltaggar som XML, och därför måste dessa har sluttaggar (end tags). Om du inte vill skriva in en “end tag” som i till exempel

```
<img src=”myphoto.jpg” runat=”server”></img>
```

kodar du bara in en slash innan du skriver slutparentesen, så här

```
<img src=”myphoto.jpg” runat=”server” />
```

5. När körningen av alla serverkontroller avslutas (det vill säga när hela uppsättningen med serverkontroller är färdigkörda) kommer ASP.NET i tur och ordning instruera respektive kontroll att rendera (det vill säga att skapa och förse besökaren med rätt HTML-kod).
   - Om originaltaggen bestod av vanlig HTML-kod skickar ASP.NET den utan förändring.
   - Om originaltaggen skapade en serverkontroll, anropar ASP.NET kontrollens *Render*-metod för att generera den HTML-kod besökaren ska erhålla.
6. När koden för sidan har lämnat webbservern, lämnar ASP.NET serverkontrollerna och övriga komponenter som kan ha skapats.

En hierarki av kontroller

När ASP.NET laddar serverkontrollerna till minnet, lagras de inte som en "platt" lista. Dessa lagras i stället som ett hierarkiskt träd. Detta tillvägagångssätt är ganska fyndigt, och illustrerar styrkan i ASP.NET.

Varje serverkontroll har möjlighet att innehålla ytterligare serverkontroller. Det är faktiskt så att ASP.NET-sidan i sig själv är en serverkontroll, en *Page server control*. Kontrollen *Page* använder sin kontrollsamling (*Controls* collection) för att spara en lista med ytterligare serverkontroller. Var och en av dessa kontroller har i sin tur en kontrollsamling som kan spara ännu flera kontroller – och så vidare.

Denna hierarki av kontroller är användbar i situationer som visas i nedanstående exempel. En <*table*>-tagg innehåller två <*tr*>-taggar, och var och en av <*tr*>-taggarna innehåller <*td*>-taggar.

```
<table id="tblPlenty" runat="server">
  <tr id="rowPlentyHeading" runat="server">
    <td colspan="3" id="celPlentyTitle" runat="server"></td>
  </tr>
  <tr id="rowPlentyDetail" runat="server">
    <td id="celPlentyAmount" runat="server"></td>
    <td id="celPlentyUnit" runat="server"></td>
    <td id="celPlentyDescription" runat="server"></td>
  </tr>
</table>
```

Anta att all denna kod fanns inom en webbsida. Toppen på hierarkin skulle utgöra en *Page*-kontroll. Utifrån denna skulle:

- Kontrollsamlingen i serverkontrollen Page innehålla serverkontrollen *HtmlTable* med namnet *tblPlenty*.
- Kontrollsamlingen i serverkontrollen HtmlTable innehålla två serverkontroller *HtmlTableRow* med namnen: *rowPlentyHeading* och *rowPlentyDetail*.
- Serverkontrollen *HtmlTableRow* med namnet *rowPlentyHeading* skulle ha en kontroll i sin kontrollsamling: serverkontrollen *HtmlTableCell* med namnet *celPlentyTitle*.
- Serverkontrollen *HtmlTableRow* med namnet *rowPlentyDetail* skulle ha tre delar i sin kontrollsamling: serverkontrollerna *HtmlTableCell* med namnen *celPlentyAmount*, *celPlentyUnit* och *celPlentyDescription*.

När ASP.NET instruerar sidan att rendera sig själv så:

- Instruerar kontrollen *Page* varje medlem i sin kontrollsamling att rendera sig själv vid rätt tidpunkt. I detta exempel skulle kontrollen *Page* instruera serverkontrollen *tblPlenty* att rendera sig själv.
- Varje kontroll (som till exempel *tblPlenty*) som har en underordnad kontroll instruerar dessa underordnade kontroller att rendera sig själva. Till exempel skulle kontrollen *tblPlenty* instruera kontrollerna *rowPlentyHeading* och *rowPlentyDetail* att rendera sig själva.
- I likhet med detta skulle *rowPlentyHeading* instruera kontrollen *celPlentyTitle* att rendera sig själv och slutligen skulle kontrollen *rowPlentyDetail* instruera kontrollerna *celPlentyAmount*, *celPlentyUnit* och *celPlentyDescription* att rendera sig själva.

Programkod kan manipulera serverkontroller, skapa nya och radera de som inte passar för ett visst ändamål. Till exempel, efter att ha hämtat en rad från en databasfråga, kan man med programkod skapa ett objekt *HtmlTableRow*, lägga till ett objekt *HtmlTableCell* för att visa respektive output-fält och därefter lägga till ett objekt *HtmlTableRow* till en befintlig tabell.

Hantera händelser

Serverkontroller i ASP.NET lever ett flyktigt liv. Hela livscykeln med att läsa, köra och sända iväg en ASP.NET-sida går på några bråkdelar av en sekund: millisekunder eller mindre.

Men under denna bråkdels sekund bombarderar ASP.NET varje serverkontroll med en serie meddelanden som kallas *händelser* (events). För varje händelse, kommer en given serverkontroll ha, eller inte ha, en matchande *händelsehanterare* (event handler) – vilket är en slags funktion eller subrutin. Om det finns en händelsehanterare kommer ASP.NET att aktivera denna då en viss händelse uppstår.

ASP.NET aktiverar upp till 30 händelser för varje serverkontroll på en sida. Dessa händelser behöver du vanligtvis inte bry dig om som programmerare, oavsett hur skicklig du är. I tabell 2-1 listas de händelser som en webbutvecklare använder mest.

Tabell 2-1
Serverkontrollernas vanligaste händelser.

Händelse	Uppstår då
OnInit	ASP.NET initialiserar en serverkontroll.
OnLoad	ASP.NET avslutar laddningen av en serverkontroll.
OnPreRender	ASP.NET är redo att initiera förfrågningar till serverkontroller att rendera sig själva.
OnUnload	ASP.NET är i färd med att radera serverkontrollen från minnet.

Bland dessa händelser är händelsen *OnLoad* för objektet *Page* antagligen den mest användbara. Denna händelse inträffar när ASP.NET har slutfört laddningen av alla sidans kontroller till minnet, men innan ASP.NET har börjat skicka webbsidan till besökaren. Detta är vanligtvis ett perfekt tillfälle att genomföra de olika typer av bearbetning som sidan kräver.

Vissa serverkontroller i ASP.NET genererar serverbaserade händelser som ett svar på händelser som sker i webbläsaren. Om användaren till exempel klickar på en knapp, eller ändrar ett alternativ i en listruta kan detta:

- Initiera en förfrågan till webbservern.
- Köra om koden på samma sida.

- Sätta igång en specialhändelse som återspeglar en knappklickning eller ett nytt urval.
- Köra en skräddarsydd händelsehanterare som utför de processer som händelsen kräver.

Det faktum att ASP.NET kopplar varje signifikant webbläsarhändelse med olika händelsehanterare på serversidan är riktigt bra. Det gör det mycket enklare att säkerställa att rätt kod körs vid varje händelse.

Genomgång av en sidas livscykel

Detta avsnitt introducerar livscykeln över en ASP.NET-sida under det att den passerar en webbserver. I detalj händer följande:

- ASP.NET börjar med att begära sidan från serverns filsystem och laddar denna till minnet. Om en HTML-tagg innehåller attributet *runat="server"* laddar ASP.NET motsvarande serverkontroll. I annat fall, sparar den taggen som vanlig HTML och skickar den vidare oförändrad.
- Programkod i serverkontrollen körs beroende på vilka händelser som initieras, som till exempel *OnLoad* och händelser från formulärfält. Denna programkod kan komma åt resurser på serversidan och ändra egenskaperna för serverkontrollerna som det passar.
- När alla händelsehanterare för serverkontrollerna har avslutats, skickar ASP.NET en förfrågan till respektive kontroll att rendera sig själv och skicka resultatet (i sekvens) till webbesökaren.
  - Konventionella HTML-taggar skickas vidare till besökaren helt oförändrade.
  - Om en serverkontroll är inblandad, skickar ASP.NET inte den XML-kod som laddade kontrollen. I stället genererar kontrollens *Render*-metod HTML-koden och/eller andra data som skickas vidare till besökaren.
- Så snart ASP.NET har skickat hela sidan till besökaren, överges alla objekt som representerade sidan.

Kategorisera serverkontroller i ASP.NET

Som de flesta saker här i livet, finns det flera alternativa sätt att kategorisera serverkontroller i ASP.NET. Tekniskt sett är kategorierna följande:

- **Web Custom Controls** (anpassade serverkontroller) finns helt och hållet inom en viss DLL. Flera projekt eller webbplatser kan därför använda en enkel kopia av kontrollen. Detta ger en bra försäkring om att samma kod körs i vart och ett av dessa projekt. Men, för att ändra kontrollen behöver du lokalisera programmets originalkod, ändra den, kompilera om den och sedan ersätta alla förekomster av den DLL som detta resulterar i.
- **Web User Controls** (användarkontroller) består av en .ascx-fil (som innehåller fragment av HTML-kod) och antingen källkodsfiler eller en DLL. Varje projekt eller webbplats som använder en användarkontroll (user control) måste ha sin egen utgåva av dessa filer. Detta gör att användarkontroller är lättare att utveckla och ändra, men svårare att hålla synkroniserade mellan flera projekt.

Om du utvecklar en serverkontroll för ASP.NET är det rent praktiskt antagligen bäst att utveckla en användarkontroll (user control). Om du utvecklar en kontroll för användning i flera projekt, eller en kontroll för försäljning – så är en anpassad serverkontroll (custom control) antagligen ett bättre val.

ASP.NET tillhandahåller ett rejält utbud av serverkontroller. De är samtliga Web custom controls och dessa kan delas upp i ytterligare kategorier:

- **HTML Server Controls** kopierar syntax och funktion från traditionella HTML-taggar. När du jobbar med traditionella HTML-taggar och lägger till *runat="server"*, skapar du en HTML-serverkontroll.

  Nyblivna ASP.NET-utvecklare brukar uppskatta dessa kontroller då syntaxen är bekant och att program för webbdesign, som till exempel Microsoft FrontPage®, kan visa hur dessa ser ut i grafiskt editeringsläge.

- **Web Server Controls** är mer kraftfulla än HTML-serverkontroller, men använder en helt annorlunda syntax.

Anpassa en användarkontroll eller använda en anpassad serverkontroll?

Många utvecklare har problem med att skilja mellan begreppen anpassad serverkontroll (Web custom control) och användarkontroll (Web user control). För att klargöra hur dessa skiljer sig åt, kom ihåg att:

- *Anpassade serverkontroller löser behovet för många användare och finns i en DLL.*
- *Användarkontroller är enklare för individuella webbutvecklare (användare) att skapa och ändra.*

SE ÄVEN

I kapitel 8 förklaras hur du skapar och använder dina egna användarkontroller (Web user controls).

Följande kod skapar en *HtmlSelect*-kontroll som visar en listruta med världens kontinenter.

```
<select id="selContinent" runat="server">
    <option value="AF">Africa</option>
    <option value="AS">Asia</option>
    <option value="AU">Australia</option>
    <option value="EU">Europe</option>
    <option value="NA">North America</option>
    <option value="SA">South America</option>
</select>
```

I nästföljande kodblock skapas serverkontrollen *DropDownList* som renderar en identisk listruta.

```
<asp:DropDownList id="ddlContinent" runat="server">
    <asp:ListItem Value="AF">Africa</asp:ListItem>
    <asp:ListItem Value="AS">Asia</asp:ListItem>
    <asp:ListItem Value="AU">Australia</asp:ListItem>
    <asp:ListItem Value="EU">Europe</asp:ListItem>
    <asp:ListItem Value="NA">North America</asp:ListItem>
    <asp:ListItem Value="SA">South America</asp:ListItem>
</asp:DropDownList>
```

Så varför ska man använda en mer komplex och okänd webbserverkontroll? Ett vanligt skäl är att det endast är serverkontrollen DropDownList som kan generera händelser på servern. Du kan till exempel lägga till attributen som visas i grönt nedan till ovanstående tagg:

```
<asp:DropDownList id="ddlContinent" AutoPostBack="True"
OnSelectedIndexChanged="ddlContinent_SelectedIndexChanged"
runat="server">
```

- Attributet *AutoPostBack="True"* instruerar kontrollen att generera JavaScript-kod som omedelbart skickar sidan till webbservern så snart besökaren väljer en annan kontinent.
- Attributet *OnSelectedIndexChanged* instruerar kontrollen att köra en händelsehanterare med namnet *ddlContinent_SelectedIndexChanged* så snart sidan skickas ut till webbläsaren och kommer tillbaka med en annan kontinent vald.

Denna händelsehanterare kan till exempel ladda om ytterligare en DropDownList-kontroll med namnet på de länder som finns inom denna kontinent.

De mest kraftfulla serverkontrollerna överger all likhet med konventionella HTML-taggar. De skapar klickbara kalendrar, listrutor eller dynamiska menyer, webbplatsöversikt, länkstigar, loginknappar, formulär för registrering och en stor mängd fördefinerade och användbara objekt. Samtliga dessa renderar sig själva som vanlig HTML-kod och fungerar därför i alla webbläsare.

Summering...

ASP.NET kör webbsidor genom att ladda dem in till minnet som en samling vanliga HTML-segment och serverkontroller. Det aktiverar händelser i varje serverkontroll, vilket får angivna funktioner eller subrutiner (händelsehanterare) att köras. Slutligen erhåller ASP.NET den HTML-kod som ska visas för besökaren genom att anropa respektive kontrolls Render-metod.

Det finns två huvudsakliga typer av serverkontroller: användarkontroller (Web user controls) som är enklare att utveckla och anpassade serverkontroller (Web custom controls) som lämpar sig bättre för bredare programspridning.

ASP.NET levereras med två olika typer av inbyggda serverkontroller. HTML-serverkontroller påminner om konventionella HTML-taggar i sin form och funktion. Web-serverkontroller kräver taggar som skiljer sig från vanlig HTML-kod, men innehåller många specialfunktioner och skickar vanlig HTML-kod till webbläsaren. Tekniskt sett, utgör både HTML-serverkontroller och Web-serverkontroller så kallade anpassade serverkontroller (Web custom controls).

Nästa kapitel förklarar hur du arbetar med all den programkod som din ASP.NET-sida kräver.

TIPS

I kapitel 8 förklaras hur du lägger till och konfigurerar såväl HTML-serverkontroller och Web-serverkontroller på alla ASP.NET-sidor.