ASP.NET 2.0 (Whidbey)

Mi várható a 2005-ös ASP.NET-ben?

VII. rész: Hierarchikus adatok kezelése

Sorozatunk korábbi részeiben már szó volt a Data Source vezérlőkről, és az adatkötés újdonságairól. Ebben a cikkben kiterjesztjük az adatkötést hierarchikus adatokra is, megnézzük azok kezelését és megjelenítését. Ez nem olyan egyszerű, mint a relációs módszer.

Motiváció

Az ASP.NET 1.x vezérlői nem támogatták hierarchikus adatok kezelését, pedig egy webalkalmazásban sokszor kell nem-relációs adatokkal dolgozni. Menük, site-térképek, valamely szempont szerint csoportosított termékek, termékosztályok, alkatrészjegyzékek, cégek szervezeti listája és számtalan más területen nem relációs, hanem hierarchikus adatokat kell megjeleníteni és kezelni.

Megjelenítés terén az Internet Explorer Web Controlok [1] adtak némi támogatást, mert abban létezik egy `TreeView` vezérlő, ami XML tartalmat képes megjeleníteni egy fában. Érzésem szerint ennek ügyféloldali kódja még az Outlook Web Accesshez kifejlesztett kódjából származik. Ezt továbbfejlesztették az ASP.NET 2.0-ban, ebből lett a `TreeView` vezérlő, amit hamarosan kielemzünk. Emellett a `Menu` és a `SiteMapPath` vezérlők képesek hierarchikus adatmegjelenítésre, ezekkel a cikksorozat következő része fog foglalkozni.

Ha vannak új megjelenítő vezérlők, akkor az ASP.NET 2.0 filozófiájának megfelelően Data Source vezérlők is kellenek hozzájuk. Irány az `XmlDataSource`!

Egyszerű hierarchikus adatmegjelenítés

Mielőtt belemélyednénk a részletekbe, nézzünk egy példát a vezérlő használatára. Hogy lássunk is belőle valamit, hozzákötöttem egy `TreeView`-t.

A forrás xml-ünk:

```
<?xml version="1.0" encoding="utf-8" ?>
<bookstore>
  <genre name="Living">
    <book ISBN="BU1032"
      title="Why Do Men Have Nipples?"
      Price="10.99">
      <chapter num="1" name=
       "How do people in wheelchairs have sex?">
        Abstract...
      </chapter>
      <chapter num="2" name="Why do old people
grow hair on their ears?">
        Abstract...
      </chapter>
      <chapter num="3"
        name="Why do I get a killer headache when
I suck down my milkshake too fast?">
        Abstract...
      </chapter>
    </book>
    <book ISBN="BU2075"
          title="You Can Combat Computer Stress!"
          Price="2.99">
...
    </book>
  </genre>
</bookstore>
```

1. lista: attribútumalapú xml adatforrás

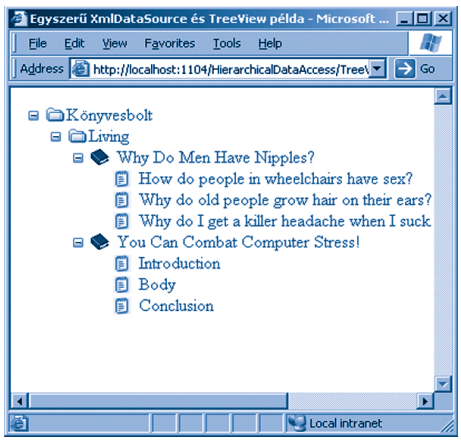
A megjelenítő webform kódja:

```
<%@ Page Language="C#" %>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Egyszerű XmlDataSource
    és TreeView példa</title>
</head>
<body>
  <form id="Form1" runat="server">
    <asp:XmlDataSource ID="MySource"
      DataFile="~/App_Data/BooksANF.xml"
      runat="server" />
    <asp:TreeView ID="TreeView1"
      DataSourceID="MySource"
      ExpandDepth="3" MaxDataBindDepth="3"
      runat="server">
      <DataBindings>
        <asp:TreeNodeBinding
```

```
          Text="Könyvesbolt"
          ImageUrl="images/folder.gif" />
        <asp:TreeNodeBinding
          DataMember="genre"
          TextField="name"
          ImageUrl="images/folder.gif" />
        <asp:TreeNodeBinding
          DataMember="book" TextField="title"
          ImageUrl="images/closedbook.gif" />
        <asp:TreeNodeBinding
          DataMember="chapter"
          TextField="name"
          ImageUrl="images/notepad.gif" />
      </DataBindings>
    </asp:TreeView>
  </form>
</body>
</html>
```

2. lista: adatkötés TreeView-hoz

A végeredmény:



1. ábra: TreeView XmlDataSource adatforrással

Az `XmlDataSource.DataFile` jellemző egy XML dokumentum elérési útját várja paraméterül. Ha az XML tartalom már stringként rendelkezésre áll, akkor a `Data` jellemzőn keresztül az is megadható forrásként.

Transzformáció XSLT-vel

Ha a forrás XML nem a megfelelő formátumú, akkor egy XSLT transzformáció is futtatható az adatokra, amelyet fájl elérési útként a `TransformFile` jellemzőben adhatunk át neki. Ez is betölthető stringként, a `Transform` jellemzőn keresztül. Ha a transzformáció paraméterezhető (a'la `xsl:param`), akkor paramétereket a `TransformArgumentList` jellemző használatával passzolhatunk át neki. Miért lehet szükség transzformációra, hisz az `XmlDataSource`-nek teljesen érdektelen a forrásadatok felépítése? Neki igen, de a megjelenítő vezérlőknek már nem. Például a `TreeView` attribútumok és nem gyermekelemek tartalmát szereti feldolgozni, így egy elemalapú leképezést használó XML dokumentumot nehéz vele megjeleníteni. Az alábbi dokumentum például az előbbi példa részben elemalapú megfelelője:

```
<bookstore>
  <genre name="Living">
    <book>
      <ISBN>BU1032</ISBN>
      <title>Why Do Men Have Nipples?</title>
      <price>19.99</price>
      <chapters>...
    <book>
      <ISBN>BU7832</ISBN>
      <title>Straight Talk About Computers</title>
      <price>19.99</price>
    </book>
  </genre>
</bookstore>
```

3. lista: elemalapú xml adatforrás

Az `ISBN` és társaik lekerültek gyermekelemekbe. Ennek nem örülne a `TreeView`, ezért átalakítjuk a korábbi formátumra egy XSLT-vel:

```
<?xml version="1.0" encoding="utf-8" ?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- Identitás transzformáció -->
  <xsl:template match="/ | node() | @*">
    <xsl:copy>
      <xsl:apply-templates select="@* | node()"/>
    </xsl:copy>
  </xsl:template>

  <!-- A book elemet speciálisan le
  kell kezelni, az elemről
  attributumra transzformáció miatt. -->
  <xsl:template match="book">
    <xsl:copy>
      <xsl:for-each select="*">
        <xsl:choose>
          <xsl:when test="count(child::*) = 0">
            <xsl:attribute name="{name()}">
              <xsl:value-of select="."/>
            </xsl:attribute>
          </xsl:when>
          <xsl:otherwise>
            <xsl:apply-templates select="./*"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```

4. lista: XSLT transzformáció a gyermekelemek attribútumokká átfordításához

A transzformáció elég trükkös, megér pár szót. Cél a `book` elemek gyermekelemeinek átfordítása attribútumokká. Az összes többi tartalom egy az egyben mehet bele a kimenetbe. A felső részben látható Identitás transzformáció

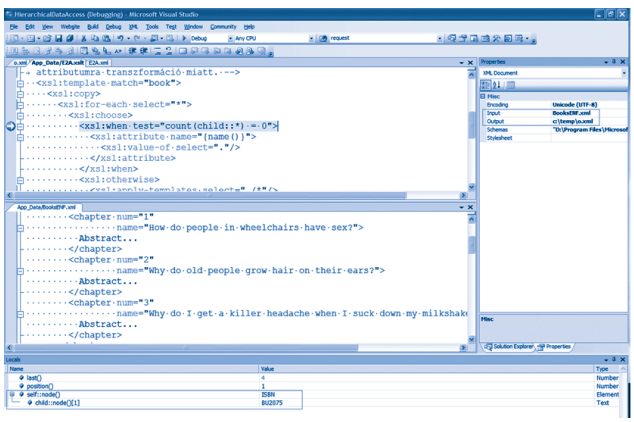
feldolgozza a dokumentumot (/), annak összes csomópontját (`node()`) (elem, szöveges tartalom, megjegyzés, vezérlő utasítás), és attribútumát (`@*`). Az `xsl:copy` mindegyiket átmásolja a kimenetbe, az `xsl:apply-templates` pedig rekurzívan végighajtja ezt a sablont az összes csomóponton és attribútumon. Ez önmagában ugyanazt rakná ki a kimenetre, mint a forrás xml, ezért hívják Identitás transzformációnak. A második sablon viszont belekotyog a másolásba. Amikor a fenti `xsl:apply-templates` feldob egy `book` elemet feldolgozásra, akkor két sablon is jelentkezik, hogy ért hozzá, az első a `node()` miatt, a második a `book` XPath kifejezés miatt. Az XSLT szabályai szerint mivel a második specifikusabb, ezért az fut le. Azaz a `book` elemeket *nem* az Identitás transzformáció dolgozza fel, hanem a második sablon.

Ebben minden gyermekelemet érintő másolás történik (vesd össze az elemalapú XML doksival, 1. lista), de úgy, hogy az elemekből attribútumokat gyártunk, de akkor, és csakis akkor, ha levélszintű elemről van szó, azaz olyanról, amelynek nincs gyermekeleme, másképpen megfogalmazva a gyermekelemek száma 0. Ezt a cikornyás feltételt fogalmazza meg a `count(child::*) = 0` kifejezés.

Miért kell ez? Mert a `chapters` elemet és gyermekeit nem akarjuk bántani, azokat egy az egyben tovább akarjuk másolni. Rájuk az `xsl:otherwise` ág fut le, ahol minden gyermekelemet (`./*`) felajánlunk másolásra. Azért a gyermekelemeket, mert ebben a forrásban a `chapter` elemeket egy `chapters` elem tartalmazza, és fel akarjuk emelni a `chapter` elemeket a `book` alá, mint a kiinduló attribútum alapú példában.

A `chapter` elemek másolását már az Identitás trafó intézi. A példa annyiból nem tökéletes, hogy ha a book alatt lenne megjegyzés vagy vezérlőparancs, akkor azt nem másolja át a kimenetre. Ehhez az `<xsl:for-each select="*">` -ot át kellene írni `<xsl:for-each select="node()">`-ra, ám az így belépő nem-elem csomópontokat külön le kellene kezelni `xsl:if`-fel vagy `xsl:when`-nel, ami feleslegesen megnehezítené a példa megértését.

Hogyan lehet egy ilyen trükkösebb XSLT-t tesztelni? A VS 2005-ben már van XSLT debugger, ami nagy kincs ám! Ennek használatához be kell tölteni az XSLT-t az IDE-be, valamit beállítani neki valamilyen forrás XML-t. Ezek után a szokásos módon F9-cel töréspontokat rakhatunk az XSLT-be, majd az XML menü Debug XSLT menüpontjával elindul a játék.



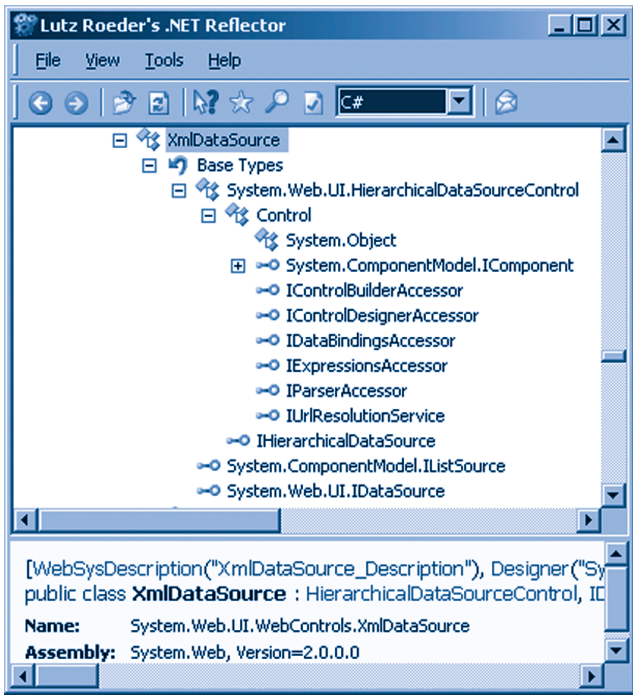
2. ábra: VS 2005 XSLT debugger

De most már végre használjuk is fel a transzformációnkat:

```
<asp:XmlDataSource ID="MySource"
  runat="server"
  DataFile="~/App_Data/BooksENF.xml"
  TransformFile="~/App_Data/E2A.xslt" />
```

Az XmlDataSource

Minden hierarchikus adatforrás, így az XmlDataSource is az `IHierarchicalDataSource`-t implementálja:



3. ábra: az XmlDataSource osztályhierarchiája a Reflectorban

Az `IHierarchicalDataSource` így néz ki:

```
public interface IHierarchicalDataSource {
  event EventHandler DataSourceChanged;
  HierarchicalDataSourceView
    GetHierarchicalView(string viewPath);
}
```

A megjelenítő vezérlő (mint az előbbi példákban a `TreeView`) a `GetHierarchicalView` metódus hívásával lekér egy `HierarchicalDataSourceView` implementációt az `XmlData-Source`-tól.

```
public abstract class HierarchicalDataSourceView {
  public abstract IHierarchicalEnumerable
    Select();
}
```

Ezen egyetlen `Select` metódus ül, ami egy hierarchikusan bejárható példányt ad vissza, az `XmlDataSource` egy `XmlHierarchicalDataSourceView`-t.

Az `IHierarchicalEnumerable` leszármazik a normál adatok bejárását támogató `IEnumerable` interfészből.

```
public interface IHierarchicalEnumerable :
   IEnumerable {
   IHierarchyData GetHierarchyData(
     object enumeratedItem);
}
```

Emiatt bejárható `foreach`-csel, amiben a visszakapott elmek a dokumentum gyökérelemei. Azért lehet több gyökér, mert XML dokumentum-töredékek is feldolgozhatók, és mert egy induló kiválasztó XPath feltétellel leválogatható a dokumentum egy részhalmazát alkotó node-list is (`viewPath` paraméter az `IHierarchicalDataSource.GetHierarchicalView`-ban)).
A bejárás során `IHierarchyData` implementációkat kapunk vissza, amellyel tovább lehet fúrni a hierarchiában.

```
public interface IHierarchyData
{
      IHierarchicalEnumerable GetChildren();
      IHierarchyData GetParent();

      bool HasChildren { get; }
      object Item { get; } //XmlNode
      string Path { get; }
      string Type { get; } //Név
}
```

Az `XmlDataSource`-hoz az `XmlHierarchyData` implementálja az `IHierarchyData` interfészt. Ez az osztály már internal láthatóságú, mint általában az iterátorok, így csak a fenti interfészen keresztül érhető el.
De Reflectorban [2] jól látszik, hogy belül egy XmlNode-ra tart egy referenciát, abból vezeti le a fenti metódusokat. Pl. a `HasChildren` implementációja:

```
bool HasChildren {
  get {
    return this._item.HasChildNodes;
  }
}
```

Hogy az egész eddigi barangolásunknak értelmet adjunk, írjunk egy egyszerű programot, ami kézzel bejárja az előző XML dokumentumok egyikét, és abból fát épít.
Kiindulási állapotunk ez lesz:

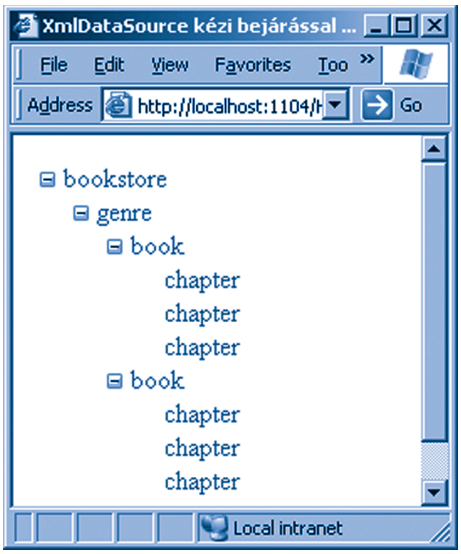
```
<asp:XmlDataSource ID="MySource"
  DataFile="~/App_Data/BooksANF.xml"
  runat="server" />
<asp:TreeView ID="TreeView1"
  runat="server">
</asp:TreeView>
```

A háttérkód, sok megjegyzéssel fűszerezve:

```
protected void Page_Load(object sender, EventArgs
e) {
  IHierarchicalDataSource dataSource =
(IHierarchicalDataSource)MySource;

  //Lekérünk egy nézetet a
  //gyökérszinttől kezdve
  HierarchicalDataSourceView view =
    dataSource.GetHierarchicalView("/");
  //Legyártatjuk a hierachikusan bejárható
  //kollekciót
  IHierarchicalEnumerable rootData =
    view.Select();
  //Elindul a faépítés
  BuildTreeRecursive(rootData,
    TreeView1, null, 0);
}

private void BuildTreeRecursive(
  IHierarchicalEnumerable parentData,
  TreeView view, TreeNode node, int depth)
{
  //A forrás lehet xml töredék is,
  //azaz több gyökéreleme is lehet
  foreach (IHierarchyData data in parentData)
  {
    TreeNode newNode = new TreeNode();
    newNode.Text = data.Type;

    if (depth == 0)
    {
      //Gyökércsomópont, közvetlenül
      //a fába megy
      view.Nodes.Add(newNode);
    }
    else
    {
      //Gyermekcsomópont lesz, a
      //kapott szülőhöz (előző szint)
      //fűzzük hozzá
      node.ChildNodes.Add(newNode);
    }
    //Lekérjük a kölyköket
    IHierarchicalEnumerable childNodes =
      data.GetChildren();
    if (childNodes != null)
    {
      //Kezdjük az egészet elölről
      BuildTreeRecursive(childNodes, view,
newNode, depth + 1);
    }
  }
}
```

5. lista: programozott hierarchikus adatforrás bejárás és megjelenítés

Végeredmény:



4. ábra: fafelépítés programozottan

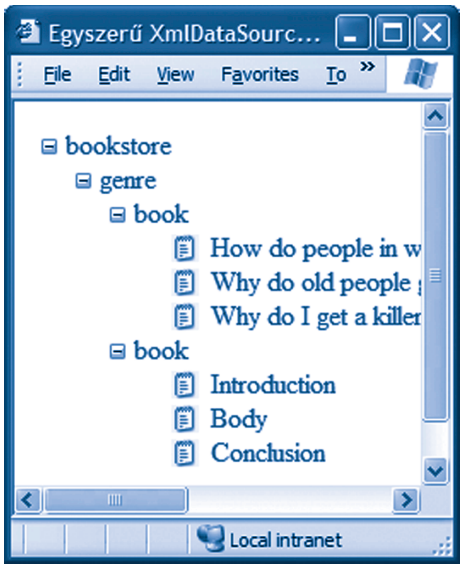
Ez persze még elég butácska, de legalább sikerült kibontani a hierarchikus forrásadatokat.
A `newNode.Text = data.Type;` sor az, amin még sokat lehetne finomítani, amit meg is teszünk a következőkben.

TreeView adatkötéssel

A `TreeView` képes automatikusan felépíteni a tartalmát valamely hierarchikus adatforrásból.
Ha csak egyszerűen kap egy adatforrást, de nem segítünk neki semmit, akkor pont azt a képet láthatjuk, mint a 4. ábrán: kiírja az elemek nevét.
Az adatok leképezését a fa csomópontjaira `TreeNodeBinding` objektumok segítségével szabályozhatjuk. Például:

```
<asp:TreeNodeBinding
 DataMember="chapter"
 TextField="name" />
```

hatására a fa kimenete az 5. ábrán látható lesz.
Ez azt írja elő, hogy miközben a vezérlő rekurzívan bejárja az adatforrást, és találkozik `chapter` nevű elemmel, akkor annak `name` attribútumát jelenítse meg az adott hierarchiaszinten (és ne az elem nevét, mint a binding nélküli többi elemnél).
Ha szeretnénk mindenhol „értelmes” tartalmat látni, akkor az összes szinthez létre kell hozni `TreeNodeBinding`-okat, hasonlóan a cikk elején látható 2. listához.
Csak attribútumokhoz tud tartalmat bind-olni a `TreeView`, gyermekelemekhez nem, mert az elemek határozzák meg a hierarchiát, így a gyermekelemeket mindenképpen új szinten mutatja meg. Kivéve, ha az eredeti DataBinding mechanizmust kicseréljük sajátra. A [3] címen található példakódban `SuperTreeView` (szerény) néven található egy `TreeView` leszármazott, ami kézzel építi fel a hierarchiát az 5. listának megfelelő módon, csak jelentősen kifinomultabban. Ezt terjedelmi okokból nem közlöm itt, de aki tényleg szeretné megérteni, hogyan működik a hierarchikus adatkötés, nézze meg a példakódot [3].



5. ábra: TreeNodeBinding-gal segített megjelenítés

Nemcsak a `DataMember` segítségével lehet kijelölni egy elemet, hanem a szintje segítségével is. Ha a `Depth` jellemzőt töltjük ki, az azt jelenti, hogy az adatkötés akkor indul el, ha az adott szinten érint a bejárás egy tetszőleges nevű elemet.
A kettőt kombinálni is lehet, például, előírhatunk kötést a 3. szinten levő `name` elemekre, így nem kapja fel a kötés a más szinten található name elemeket.
Emellett természetesen nem csak a csomópontok szövege köthető, hanem a navigációs url, a kép urlje, a kép tooltipje, stb., azaz minden lényeges vizuális komponens is.

Következő számunkban folytatjuk…

Soczó Zsolt
zsolt.soczo@netacademia.net
A szerző a NetAcademia vezető fejlesztőoktatója
ASP.NET MVP, MCSD, MCDBA, MCT

A cikkben szereplő URL-ek:
[1] asp.net/IEWebControls/Download.aspx
[2] aisto.com/roeder/dotnet
[3] netacademia.net/tudastar/articlepage.aspx?upid=7445