

ASP.NET 2.0 (Whidbey)

Mi várható a 2005-ös ASP.NET-ben?

VIII. RÉSZ: HIERARCHIKUS ADATOK KEZELÉSE 2/2.

Bevezetés

Az előző részben áttekintettük a hierarchikus adatkötés működését. Ebben a részben megnézzük, hogyan működik a TreeView Populate On Demand módon feltöltve, valamint megfigyeljük, hogyan lehet hierarchikus adatokat feldolgozni nemhierarchikus vezérlőkkel.

A TreeView fejlettebb szolgáltatásai

A TreeView adatforrása tekintélyes méretű csomópontot írhat le. Ügyféloldalról vizsgálva sok esetben egyáltalán nincs szükségünk az egész fa tartalmára, gondoljunk például az online msdn site tartalomjegyzékére [1]. Ezért nem feltétlenül bölcs dolog letölteni a teljes fa adatforrását, elég csak azt, ami a felhasználót érdekli. Mivel azonban előre nem ismerjük a döntését, kénytelenek vagyunk a döntése után közvetlenül letölteni az adatokat, azaz akkor, amikor kinyitja a fa valamely ágát. Ezt nevezi a TreeView Populate On Demand funkciónak. Nézzük meg közelebbről!

Populate On Demand

A szolgáltatás használatához be kell jelölni az adott Node-nál, hogy utólag lesz feltöltve (1), valamint meg kell adnunk egy Callback metódust (2), itt hívnak minket vissza adatokért:

```
<asp:TreeView ID="MyTree"
  PathSeparator="|"
  OnTreeNodePopulate="PopulateNode" (2)
  ExpandDepth="1"
  runat="server">
  <Nodes>
    <asp:TreeNode Text="PopulateOnDemandDemo"
      PopulateOnDemand="True" (1)
      Value="PopulateOnDemandDemo" />
  </Nodes>
</asp:TreeView>
```

Mint a példában látható csak a gyökérelmet hoztam létre kézzel, ennek ellenére láthatóak annak gyökérelmei is. Ennek oka, hogy az `ExpandDepth="1"` miatt rögtön meghívódik egyszer a `PopulateNode` metódusunk, ahol legeneráljuk az első szintű fájllistát (ld. 1. kép). Ez még kliensoldali visszahívás nélkül történik, gondoltak erre az esetre is a szerzők. Amikor a felhasználó rákattint valamelyik + jelre, akkor ismét meghívódik a `PopulateNode`, ám nem sima visszahívással, hanem cikksorozatunk VI. részében részletesen tárgyalt ügyféloldali visszahívások segítségével, azaz a teljes lap áll, nem frissül, csak a fa tartalma.

A `PopulateNode` feladata az új `Node`-ok felvétele a paraméterként kapott `TreeNode` objektumba, az adatok átvitelét a `TreeView` intézi `Client Callback` segítségével.

```
static readonly char slash = '/';

protected void PopulateNode(
  object source, TreeNodeEventArgs e)
{
  TreeNode node = e.Node;
  if (node.Value == "PopulateOnDemandDemo")
    node.Value = "~/";

  string rootDirectory =
    Request.MapPath(
      "~/", Request.ApplicationPath, false);
  string fullPath =
    Request.MapPath(
      node.Value, Request.ApplicationPath, false);

  //Ha ki akarnak mászni a könyvtárból...
  if (fullPath.StartsWith(
    rootDirectory,
    StringComparison.Ordinal) == false)
  {
    return;
  }

  AppendDirectoryNodes(node, fullPath);

  AppendFileNodes(node, fullPath);
}
```

Mint látható a `TreeNodeEventArgs` ad vissza referenciát arra a `Node`-ra, amely tartalmát kell kiegészíteni gyermekcsomópontokkal. A példa az alkalmazás könyvtárában található fájlokat listázza ki, az ellenőrzések azt hivatottak kiszűrni, hogy valaki hamis kéréssel kilépjen az alkalmazás könyvtárából. Nézzük az alkalmazást reprezentáló csomópontok létrehozását:

```
private static void AppendDirectoryNodes(
  TreeNode node, string fullPath)
{
  // Minden dir
  string[] dirs =
    Directory.GetDirectories(fullPath);
```

```

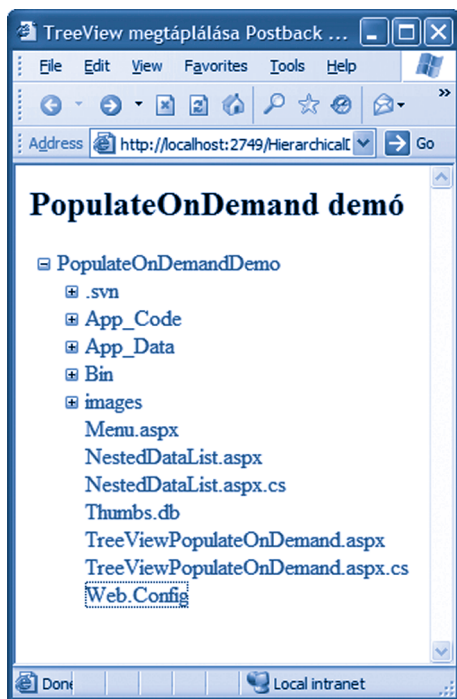
foreach (string dir in dirs)
{
    string virtualDir =
        node.Value.TrimEnd(slash)
        + "/" + Path.GetFileName(dir);

    TreeNode newNode = new TreeNode(
        Path.GetFileName(dir), virtualDir);

    // Ha még van alatta bejegyzés,
    // akkor ezt se töltjük fel előre
    if (Directory.GetFileSystemEntries(
        dir).Length > 0)
    {
        newNode.PopulateOnDemand = true;
    }
    node.ChildNodes.Add(newNode);
}
}

```

Fontos látni, hogy ha az éppen listázott könyvtár alatt vannak még bejegyzések, akkor az új Node és Populate On Demand-os kell legyen, különben nem lehetne „kinyitni” az új szintet.



1. kép: Fájlrendszert megjelenítő fa Populate On Demand-dal

A fájlok létrehozása nagyon hasonló, de itt nem kell a Populate On Demand, hisz a fájloknak nincsenek gyermek-elemeik:

```

private void AppendFileNodes(
    TreeNode node, string fullPath)
{
    // Minden fájl
    string[] files =
        Directory.GetFiles(fullPath);
    foreach (string file in files)

```

```

{
    TreeNode newNode = new TreeNode(
        Path.GetFileName(file),
        Path.GetFileName(file));
    // Csúnya, de a Path.Combine
    // nem megy URL-ekre
    newNode.NavigateUrl =
        Request.ApplicationPath + '/' +
        file.Replace(
            Request.PhysicalApplicationPath, "").
            Replace('\\', '/');
    node.ChildNodes.Add(newNode);
}
}

```

Populate On Demand a háttérben?

A fa egy ágát kinyitva a Client Callbacket elindítandó a következő kérés fut be a kiszolgálóhoz (rövidítve, dekódolva és tördelve):

```

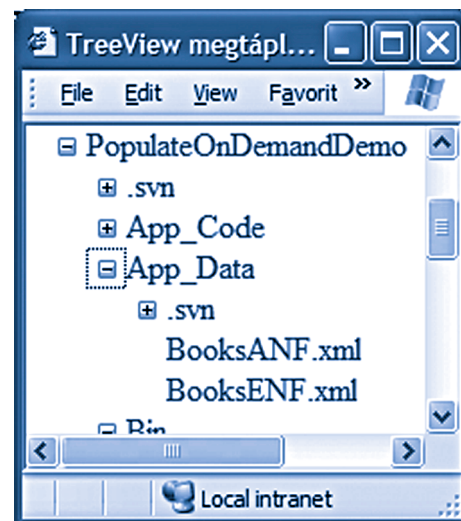
MyTree_ExpandState=eccccnnnnnnn&
MyTree_SelectedNode=&
__EVENTTARGET=&
__EVENTARGUMENT=&MyTree_PopulateLog=&
__CALLBACKID=MyTree&
__CALLBACKPARAM=3|20|fftf|8|App_Data|PopulateOnDe
mandDemo~/App_Data

```

A __CALLBACKPARAM tartalmazza azt az információt, hogy melyik ágat szeretnénk kinyitni. Én az App_Data könyvtárra kattintottam (2. kép), amely a 3. csomópont a gyökér alatt. Látható, hogy a legelső példában látható PathSeparatorban megadott karakter az elválasztó szimbólum.

A többi paraméter jelentése már csak a vezérlő kódjának visszafejtésével tudható meg, melyből kiderül például, hogy az „fftf” (false, false, true, false) jelzők arról szólnak, hogy a csomópont adatkötött-e, illetve be van-e jelölve, hisz a fa képes checkboxokat is megjeleníteni.

A paraméterek teljes visszafejtése házi feladat a kedves olvasónak, de sok új információt nem tudunk már kinyerni belőle.



2. kép: Az App_Data könyvtár került kinyitásra

Már csak egy részlettel vagyok adós: mit válaszol a kérésre a kiszolgáló? Pár egyéb infó mellett egyszerűen az új ágat megjelenítő html tartalom jön vissza:

```
<div id="MyTreen5Nodes" style="display:none;">
<table cellpadding="0" cellspacing="0"
style="border-width:0;">
<tr><td><div ...
```

Ezt aztán a lapba injektált JavaScript kód dolgozza be a fát reprezentáló DHTML struktúrába.

A JavaScriptet WebResourceként tölti le a böngésző:

```
<script
src="/HierarchicalDataAccessII/WebResource.axd?d=c
bhnwCDUBkt1FkoQ82uonA2&t=632660822442656250"
type="text/javascript"></script>
<script
src="/HierarchicalDataAccessII/WebResource.axd?d=t
xpbX1wM16aXBM6fnSz3hQ2&t=632660822442656250"
type="text/javascript"></script>
```

A WebResource szintén ASP.NET 2.0 újítás, ő egy olyan HttpHandler, amellyel erőforrásokban tárolt tartalmat lehet elküldeni a böngészőnek. Ennek előnye, hogy nem lesznek olyan problémáink, mint az 1.x-ben, amikor valamely website gyökérfástruktúrából hiányoztak az ASP.NET saját JavaScript állományai.

DataBinding nem hierarchikus vezérlőkhöz

Mit tud kezdeni egy nem hierarchikus vezérlő, például egy GridView vagy egy DataList hierarchikus adatokkal? Elsőre azt gondolnánk semmit, pedig ha belegondolunk egy gyakori feladat, a master-detail nézetek létrehozása során is hierarchikus adatokat jelenítünk meg a segítségükkel.

Igaz, ezekben a példákban a hierarchia mélysége állandó és előre ismert, de ettől még nem kevésbé hasznos funkciókról van szó.

Hogyan működnek a master-detail nézetek asp.net-ben? Általában létrehozunk egy, a master sorokat megjelenítő vezérlőt, amely minden egyes sorának generálása során létrehoz egy beágyazott másik vezérlőt, amely a master adott sorához tartozó részletező sorokat jeleníti meg.

Általában az ItemCreated vagy a DataBind eseményekben történik ez meg.

Adatforrásként az 1.x Frameworkben általában DataSet-et használtunk, és a két adatot reprezentáló DataTable-t DataRelation-nel kötöttük össze. Felfoghatjuk ezt egy kétszintű hierarchiájú rendszernek is.

Most viszont van XmlDataSource vezérlőnk, ez képes hierarchikus adatokat kezelni, ezt kellene összeházasítani például a DataList vezérlővel. Lássuk hogyan lehetséges ez! Példánkban az XmlDataSource kapott egy szűrőfeltételt (1), amely book csomópontok listáját válogatja le. Ezt kapja meg adatforrásként a DataList. Végigmegy minden csomóponton, majd meghívja rájuk a megadott XPath kifejezéseket (mint a (2)).

Ez is ASP.NET 2.0 újítás, mely segítségével Data Binding kifejezésekben (<# %>) hierarchikus adatforráson futtathatunk egy XPath kifejezést.

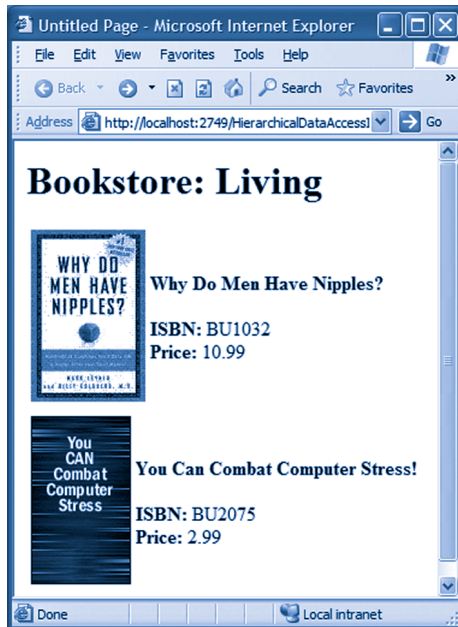
A kifejezésekben a book objektumtól, mint relatív kezdőponttól kiindulva válogatjuk le a megjelenítendő csomópontokat. A példát a 3. képen tekinthetjük meg.

```
<asp:XmlDataSource
ID="MySource"
DataFile="~/App_Data/BooksANF.xml"
XPath="bookstore/genre[@name='Living']/book" (1)
runat="server" />
<asp:DataList ID="MyDataList"
DataSourceID="MySource"
runat="server">
<ItemTemplate>
<table>
<tr>
<td>
<img src='<# "images/" +
XPath("@ISBN") + ".gif" %>'
alt="<# XPath("@title") %>" (2)
</td>
<td>
<h4>
<# XPath("@title") %>
</h4>
<b>ISBN:</b>
<# XPath("@ISBN") %>
<br>
<b>Price:</b>
<# XPath("@Price") %>
<br>
</td>
</tr>
</table>
</ItemTemplate>
</asp:DataList>
```

A példában még nem volt hierarchia kezelés, hiszen egy szintet jártunk be és jelenítettünk meg. Azonban semmi akadálya, hogy újabb DataList-et ágyazzunk be az előbbi lista ItemTemplate-jébe:

```
...
<asp:DataList ID="MyDataList"
DataSource='<# XPathSelect("chapter") %>'
runat="server">
<ItemTemplate>
<br>
<u>Chapter
<# XPath("@enum") %>
:
<# XPath("@name") %>
</u>
<br>
<# XPath(".") %>
</ItemTemplate>
</asp:DataList>
</ItemTemplate>
```

A kérdés ebben az esetben az, hogy milyen adatforrást adjunk meg listánknak, hisz a külső lista egy sorából kiindulva kell neki feldolgozni egy csomóponthalmazt? Most az XPathSelect kifejezés siet a segítségünkre.



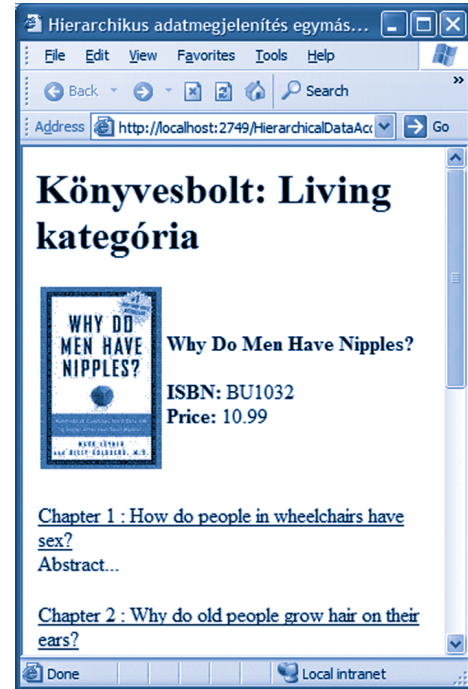
3. kép: XmlDataSource-DataList barátság

Az XPathSelect lekéri a befoglaló vezérlő, azaz a külső lista aktuálisan generált sorát a Page.GetDataItem() meghívásával, majd végrehajtja rajta a megadott XPath kifejezést, és az így kapott csomópontlistát adja vissza. Esetünkben a belső lista az egyes könyvfejezeteket kapja vissza, amelyekből a már ismert XPath databinding kifejezés veszi ki az értékeket.

Hogy az XPathSelect működését jobban megvilágítsam írtam egy egyszerű XPathSelect-hez hasonló működésű metódust, amely ebben a példában az eredetivel azonos eredményt szolgáltat:

```
protected IEnumerable SimpleXPathSelect(
    string xpath)
{
    ArrayList nodes = new ArrayList();
    XPathNavigator navigator =
        ((IXPathNavigable)GetDataItem()).
        CreateNavigator();
    XPathNodeIterator selectedNodes =
        navigator.Select(xpath);
    while (selectedNodes.MoveNext())
    {
        nodes.Add(selectedNodes.Current.Clone());
    }
    return nodes;
}
```

A funkció teljes terjedelmében az System.Web.UI.XPathBinder.Select metódusban tekinthető meg.



4. kép: hierarchikus adatok egymásba ágyazott DataList-tekkel

Zárszó

A példakódok szokás szerint a [2] címen érhetők el.

Soczó Zsolt

zsolt.socz@netacademia.net

A szerző a NetAcademia vezető fejlesztőoktatója
ASP.NET MVP, MCSD, MCDBA, MCT

A cikkben szereplő URL-ek:

[1] msdn.microsoft.com

[2] netacademia.net/tudastar/articlepage.aspx?upid=8371