media solutions

Serialisierung und Deserialisierung – Daten flexibel speichern

Autor: Thomas Bandt
Datum: 1. November 2007

Inhalt

media solutions

Ausgangslage und Lösung

Es gibt Situationen in der Praxis eines (Web-) Entwicklers, in denen er Daten flexibel speichern und präsentieren möchte. Bei mir war der Anlass für die Entwicklung der nachfolgend beschrieben Vorgehensweise ein „Protokoll“, welches vom Benutzer einer Webanwendung ausgefüllt werden muss.

Das Protokoll besteht selbst aus etwa 30 Fragen, welche wiederum unterschiedlichste Antwortmöglichkeiten zulassen. Kann man einmal einen Text oder eine Zahl eingeben (TextBox), ist bei anderen Fragen die Antwort via Multiple- oder Singlechoice (CheckBoxList vs. RadioButtonList oder DropDownList) möglich.

Das impliziert nun drei Probleme, die es zu lösen gilt:

1. Wie hält man das Formular so flexibel, dass man bei Änderungen, die vorauszusehen sind, nicht einen riesigen Rattenschwanz an Nacharbeiten generiert?
2. Wie bildet man das als Objekt vernünftig ab?
3. Wie speichere ich das vernünftig in der Datenbank?

Mögliche Lösungen mag es einige geben, mal mehr, mal weniger kompliziert. Ich verzichte an dieser Stelle auf eine Gegenüberstellung und konzentriere mich auf den Weg, für den ich mich entschieden habe: das automatisierte Binden und Speichern der Formulardaten in eine Liste, die Serialisierung dieser Liste und die anschließende Speicherung des generierten XML in einem einzigen Datenbankfeld.

Die Datenbank

Die Datenbank kann aussehen wie sie will. Zum Speichern des kompletten Formulars wird nur ein einziges Feld benötigt - das kann ein Blob, ein normales Textfeld oder das neue XML-Feld im SQL Server 2005 sein. Für mein Beispiel war mir das alles zu aufwendig – ich speichere das Profil einfach in einer XML-Datei auf dem Filesystem :-).

Das Formular

Das Formular ist ein ganz normales ASP.NET-Formular bestehend aus den normalen WebControls. Um uns die Sache einfacher zu machen wird es eingegrenzt durch ein Panel – warum und wieso, dazu gleich mehr. Später kann das Formular ohne auch nur eine einzige Zeile Code anfassen zu müssen beliebig erweitert, geändert oder sogar verkleinert werden.

```
<asp:Panel ID="ProfileDataPanel" runat="server">

    Mein Name
    <br />
    <asp:TextBox ID="Name" runat="server" />
    <br />
    <br />
    Mein Land<br />
    <asp:DropDownList ID="Country" runat="server">
        <asp:ListItem></asp:ListItem>
        <asp:ListItem Value="1">Deutschland</asp:ListItem>
        <asp:ListItem Value="2">Österreich</asp:ListItem>
        <asp:ListItem Value="3">Schweiz</asp:ListItem>
    </asp:DropDownList>
    <br />
```

media solutions

```
    <br />
    Meine Lieblingsautos<br />
    <asp:CheckBoxList ID="MyCars" runat="server">
        <asp:ListItem Value="1">Ferrari</asp:ListItem>
        <asp:ListItem Value="2">Trabant</asp:ListItem>
        <asp:ListItem Value="3">Porsche</asp:ListItem>
    </asp:CheckBoxList>

    <br />
    <asp:Button ID="Button1" runat="server" Text="Profil speichern"
        onclick="Button1_Click" />

</asp:Panel>
```

Einsammeln der Daten

Hier wird es jetzt interessant. Ich hatte gerade erwähnt, dass man beim Ändern des Formulars später keine Zeile Code schreiben muss, und dass das Formular in einem Panel eingebettet ist, hier nun der Grund:

```
ProfileAttributeCollection profile = new ProfileAttributeCollection();

foreach (Control c in ProfileDataPanel.Controls)
{
    if (c != null)
    {
        switch (c.GetType().Name)
        {
            case "DropDownList":
                DropDownList ddl = c as DropDownList;
                if (!string.IsNullOrEmpty(ddl.SelectedValue))
                    profile.Add(new ProfileAttribute(c.ID, ddl.SelectedValue));
                break;
            case "TextBox":
                TextBox tb = c as TextBox;
                if (!string.IsNullOrEmpty(tb.Text))
                    profile.Add(new ProfileAttribute(c.ID, tb.Text));
                break;
            case "CheckBoxList":
                CheckBoxList cbl = c as CheckBoxList;
                foreach (ListItem li in cbl.Items)
                {
                    if (li.Selected)
                        profile.Add(new ProfileAttribute(c.ID, li.Value));
                }
                break;
        }
    }
}

BL.Profile.SaveProfile(profile);
```

Zuerst wird ein neues Profil-Objekt erstellt, in dem die einzelnen Profileigenschaften gespeichert werden. Eine Profileigenschaft ist letztlich nichts weiter als eine Klasse mit einer Property für den Schlüssel, der Beschreibt um welche Eigenschaft es sich handelt, und einer Property für den Wert:

media solutions

```
public class ProfileAttribute
{

    public ProfileAttribute()
    {
    }

    public ProfileAttribute(string key, string value)
    {
        this.key = key;
        this.value = value;
    }

    private string key;
    private string value;

    public string Key
    {
        get { return key; }
        set { key = value; }
    }

    public string Value
    {
        get { return value; }
        set { this.value = value; }
    }

}
```

Diese Eigenschaften werden gesammelt in der ProfileAttributeCollection:

```
public class ProfileAttributeCollection : List<ProfileAttribute>
{
}
```

Das sind die zwei zentralen Geschäftsobjekte, mit denen ich hier arbeite – die können natürlich beliebig erweitert und angepasst werden. Nun zum spannenderen Teil.

spätere Eingriffe in den Code werden dadurch vermieden, dass beim Speichern des Profils alle Controls innerhalb des eingrenzenden Panels automatisch abgerufen und ausgewertet werden. Dabei wird schlicht geprüft um welchen Typ es sich handelt, ist es ein Control welches relevante Daten beinhaltet, wird dies ausgewertet: bei einer TextBox wird schlicht die Text-Eigenschaft herangezogen, bei einer CheckBoxList, die ja mehrere Antwort-Möglichkeiten bietet, wird für jede getroffene Antwort ein entsprechender Eintrag in der Profil-Collection angelegt.

Das lässt sich natürlich noch erweitern, z.B. um andere Controls oder um eine Rekursion, falls das Formular aufwendiger gestaltet und verschachtelt ist.

Serialisieren und Speichern der Daten

Haben wir nun alle Daten in der Collection zusammen, wird diese im „Business Layer“ verarbeitet, d.h. serialisiert. Serialisierung heißt in diesem Fall die Erzeugung eines XML-Markups, welches alle Informationen dieses Objektes – letztendlich als Text – enthält, die notwendig sind, um das Objekt nachher auch wieder „zurück zu wandeln“ – denn darin liegt ja der Zauber.

media solutions

```
public static void SaveProfile(ProfileAttributeCollection profile)
{
    if (profile != null && profile.Count > 0)
    {
        // Serialize
        XmlSerializer s = new XmlSerializer(typeof(ProfileAttributeCollection));
        StringWriter p = new StringWriter(new StringBuilder());
        s.Serialize(p, profile);
        // Save
        DA.Profile.UpdateProfile(p.GetStringBuilder().ToString());
    }
    else
    {
        DA.Profile.UpdateProfile(null);
    }
}
```

Das generierte XML kann übrigens so aussehen:

```
<?xml version="1.0" encoding="utf-16"?>
<ArrayOfProfileAttribute xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <ProfileAttribute>
        <Key>Name</Key>
        <Value>Thomas</Value>
    </ProfileAttribute>
    <ProfileAttribute>
        <Key>Country</Key>
        <Value>3</Value>
    </ProfileAttribute>
    <ProfileAttribute>
        <Key>MyCars</Key>
        <Value>2</Value>
    </ProfileAttribute>
    <ProfileAttribute>
        <Key>MyCars</Key>
        <Value>3</Value>
    </ProfileAttribute>
</ArrayOfProfileAttribute>
```

Enthalten ist hier der Name, die Auswahl des Landes sowie zwei ausgewählte Autos aus der Auto-Liste „MyCars“.

Wurde aus dem Objekt ein einfacher String erzeugt, der das XML-File enthält, wird dieser String nun gespeichert – wie oben schon erwähnt, geschieht das in meinem Fall der Einfachheit halber einfach durch das Speichern in einer XML-Datei.

```
public static void UpdateProfile(string profile)
{
    string filename = HttpContext.Current.Server.MapPath("~/App_Data/profile.xml");
    using(TextWriter tw = new StreamWriter(filename))
    {
        tw.WriteLine(profile);
    }
}
```

media solutions

Deserialisieren und Abrufen der Profildaten

Fangen wir unten an, beim Auslesen des XML-Strings aus der XML-Datei:

```
public static string GetProfile()
{
    string filename = HttpContext.Current.Server.MapPath("~/App_Data/profile.xml");
    using (TextReader tr = new StreamReader(filename))
    {
        return tr.ReadToEnd();
    }
}
```

Und gehen gleich weiter zum Deserialisieren:

```
public static ProfileAttributeCollection GetProfile()
{
    string source = DA.Profile.GetProfile();
    if(!string.IsNullOrEmpty(source) && source.Trim().Length > 0)
    {
        XmlSerializer s = new XmlSerializer(typeof(ProfileAttributeCollection));
        TextReader profile = new StringReader(source);
        return s.Deserialize(profile) as ProfileAttributeCollection;
    }
    return null;
}
```

Hier wird nun also aus dem gespeicherten XML wieder ein „echtes“ Objekt gemacht, welches danach sofort weiterverwendet werden kann, um etwa das Formular wieder vorzubelegen.

media solutions

Formular automatisch vorbelegen

Das tun wir dann auch mal – wir binden also die gesammelten Profildaten an die einzelnen Formularelemente.

```
protected void Page_Load(object sender, EventArgs e)
{
    // Databinding
    if (!Page.IsPostBack)
    {
        ProfileAttributeCollection profile = BL.Profile.GetProfile();
        if (profile != null && profile.Count > 0)
        {
            foreach (ProfileAttribute a in profile)
            {
                foreach (Control c in ProfileDataPanel.Controls)
                {
                    if (c != null && a.Key == c.ID)
                    {
                        switch (c.GetType().Name)
                        {
                            case "DropDownList":
                                DropDownList ddl = c as DropDownList;
                                foreach (ListItem li in ddl.Items)
                                {
                                    if (li.Value == a.Value)
                                    {
                                        li.Selected = true;
                                        break;
                                    }
                                }
                                break;
                            case "TextBox":
                                TextBox tb = c as TextBox;
                                tb.Text = a.Value;
                                break;
                            case "CheckBoxList":
                                CheckBoxList cbl = c as CheckBoxList;
                                foreach (ListItem li in cbl.Items)
                                {
                                    if (li.Value == a.Value)
                                    {
                                        li.Selected = true;
                                        break;
                                    }
                                }
                                break;
                        }
                    }
                }
            }
        }
    }
}
```

Das sieht auf den ersten Blick vielleicht etwas aufwendiger aus als das Einsammeln, letztlich ist es aber der gleiche Prozess – nur in umgekehrter Reihenfolge.

media solutions

Für jede der in der Collection befindlichen Profil-Eigenschaften wird nun in einer Schleife geschaut, ob ein passendes Formularelement zur Verfügung steht. Das funktioniert, weil die ID der Formularelemente der der Profileigenschaften entspricht.

Wurde nun beispielsweise eine Eigenschaft gestrichen oder durch eine andere ersetzt, muss man den Code nicht anpassen – dann passiert hier schlicht nichts. Sinnvollerweise kann man ihn aber so erweitern, dass man veraltete Einträge dann automatisch löscht.

Zusammenfassung

Das gezeigte und im Anhang als Download verfügbare Beispiel soll nur einen Überblick über die Möglichkeiten bieten, man kann es natürlich noch verfeinern und ausbauen. Der grundlegende Vorgang hat sich bei mir aber als praxistauglich erwiesen. Das Protokoll ist seit mehreren Monaten im Einsatz und wird erfolgreich und fleißig ausgefüllt :-).