13

Rozšíření modelu poskytovatelů

Předchozí kapitola představila model poskytovatelů v ASP.NET 3.5 a vysvětlila, jak se používá ve spojení se systémy členství a správy rolí.

Jak již bylo řečeno, tyto systémy v ASP.NET 3.5 vyžadují nějaký typ zachovávání stavu uživatele po dlouhou dobu. Jejich časový interval a požadavky na zabezpečení zachování stavu jsou větší než ty v předchozích systémech, které jen používaly objekt `Session`. ASP.NET 3.5 ihned po instalaci nabízí řadu poskytovatelů využitelných jako základní konektory a naplňujících potřeby pro ukládání dat, jež vyplývají ze správy stavu zmíněných systémů.

Poskytovatelé, kteří jsou součástí výchozí instalace rámce .NET Framework 3.5, představují nejběžnější prostředky pro ukládání dat správy stavu a fungují se všemi systémy. Stejně jako většinu věcí v .NET máte ovšem možnost si dodané poskytovatele přizpůsobit a také je rozšířit.

Tato kapitola zkoumá některé z možností rozšířením modelu poskytovatelů ASP.NET 3.5. Rovněž přibližuje několik ukázkových rozšíření modelu poskytovatelů. Nejprve se ovšem podíváme na jednodušší možnosti změny a rozšíření poskytovatelů, kteří jsou součástí výchozí instalace .NET 3.5.

Poskytovatelé jsou jednou vrstvou ve větší architektuře

Jistě si z minulé kapitoly vybavujete, že vám poskytovatelé umožňují definovat vrstvu přístupu k datům pro mnoho systémů v ASP.NET 3.5. Rovněž dovolují definovat základní logiku implementace způsobu manipulace s daty. Díky nim můžete využívat různé ovládací prvky a rozhraní API příslušných systémů jednotným způsobem bez ohledu na základní metodu datování dat poskytovatelem. Model poskytovatelů dále dovoluje jednoduše zaměnit jednoho poskytovatele za jiného, aniž by to ovlivnilo další ovládací prvky a rozhraní API, jež s poskytovatelem pracují.

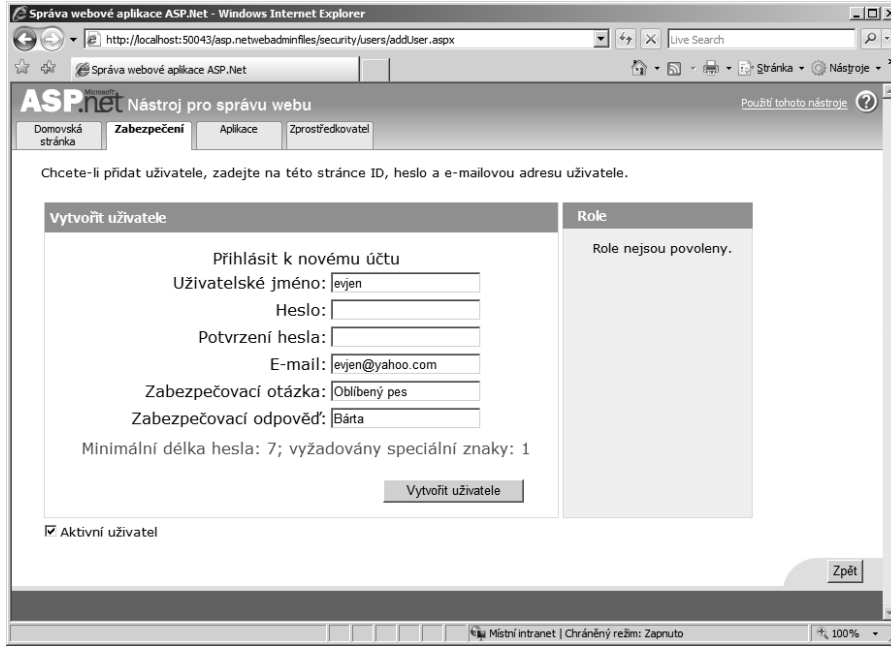
Je zřejmé, že ovládací prvky využívané systémem členství i rozhraní API `Membership` používají definovaného poskytovatele. Změna základního poskytovatele nezmění ovládací prvky ani API, ovšem v každém případě můžete upravit chování těchto položek (jak si za chvíli ukážeme). Lze také jednoduše změnit místo, kam se ukládají informace o stavu vyžadované zmíněnými položkami. Změna základního poskytovatele v tomto případě neznamená vůbec žádnou změnu pro ovládací prvky ani API; jen se jejich datové body správy stavu přesměrují na jiný typ skladu.

Změna pomocí programování atributů

Zřejmě nejjednodušší možnost změnit chování poskytovatelů vestavěných do rámce .NET Framework 3.5 představuje programování pomocí atributů. V ASP.NET 3.5 je možné prostřednictvím atributů aplikovat poměrně významné úpravy chování. Aplikovat lze jak serverové ovládací prvky, tak i nastavení v různých konfiguračních souborech aplikace. S využitím definic poskytovatelů, jak jsou v souborech `machine.config` nebo v kořenovém souboru `web.config`, můžete významně upravit chování poskytovatele. Tato kapitola ukazuje příklad úpravy poskytovatele `SqlMembershipProvider`.

Jednodušší struktury hesel díky SqlMembershipProvider

Když vytvoříte uživatele instancí poskytovatele `SqlMembershipProvider`, ať už k tomu využíváte SQL Server Express nebo Microsoft SQL Server 2000/2005/2008, všimněte si, že heslo vyžadované k vytvoření uživatele má poloviční sílu. To je zřejmé při vytváření uživatele nástrojem správy webového sídla ASP.NET, jak ukazuje obrázek 13.1.



Obrázek 13.1

Na této obrazovce jsem se pokusil zadat heslo a objevilo se upozornění, že zadané heslo nesplňuje požadavky aplikace. Objevilo se varování, že minimální délka hesla je sedm znaků a že je nutný alespoň jeden jiný než alfanumerický znak. To znamená, že je zapotřebí zadat jako heslo kupříkladu *Bubliny!*. Toto chování určuje poskytovatel členství a nikoli ovládací prvky nebo rozhraní API využívané systémem členství. Definici požadavků najdete v souboru `machine.config.comments` v adresáři `C:\WINDOWS\Microsoft.NEt\Framework\v20.50727\CONFIG`. Odpovídající zadání je ve výpisu 13.1.

Výpis 13.1: Deklarace instance SqlMembershipProvider

```
<configuration>
  <system.web>
    <membership defaultProvider="AspNetSqlMembershipProvider"
      userIsOnlineTimeWindow="15" hashAlgorithmType="" >
      <providers>
        <clear />
        <add connectionStringName="LocalSqlServer"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          applicationName="/"
          requiresUniqueEmail="false"
          passwordFormat="Hashed"
          maxInvalidPasswordAttempts="5"
          minRequiredPasswordLength="7"
          minRequiredNonalphanumericCharacters="1"
          passwordAttemptWindow="10"
          passwordStrengthRegularExpression=""
          name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider,
          System.Web, Version=2.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a" />
      </providers>
    </membership>
  </system.web>
</configuration>
```

Podíváte-li se na atributy poskytovatele, jistě si uvědomíte, že výše zmíněné chování definují atributy `minRequiredPasswordLength` a `minRequiredNonalphanumericCharacters`. Chování změníte ve všech aplikacích na serveru prostou změnou hodnot v uvedeném souboru. Doporučuji ovšem překrýt příslušné údaje v souboru `web.config` konkrétní aplikace, jak to zachycuje výpis 13.2.

Výpis 13.2: Změna hodnot atributů v souboru web.config

```
<configuration>

  <system.web>

    <authentication mode="Forms" />
    <membership>
      <providers>
        <clear />
```

```
        <add name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider,
            System.Web, Version=2.0.0.0, Culture=neutral,
          PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="LocalSqlServer"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          requiresUniqueEmail="false"
          passwordFormat="Hashed"
          maxInvalidPasswordAttempts="5"</b>
          minRequiredPasswordLength="4"
          minRequiredNonalphanumericCharacters="0"
          passwordAttemptWindow="10" />
      </providers>
    </membership>

  </system.web>

</configuration>
```

V tomto příkladu se požadavky na heslo mění atributy `minRequiredPasswordLength` a `minRequiredNonalphanumericCharacters`. Jako minimální délka hesla jsou požadovány čtyři znaky, z nichž žádný nemusí být jiný než alfanumerický (tedy kupříkladu speciální znak jako `!`, `$` nebo `#`).

Změna definice poskytovatele v souboru `web.config` aplikace je velmi snadná. Z příkladu ve výpisu 13.2 je vidět, že se element `<membership>` velmi podobá stejnému elementu v souboru `machine.config`.

Při definování své vlastní instance `SqlMembershipProvider` máte několik možností. Jedním z přístupů, jak to ukazuje výpis 13.2, je předefinovat pojmenovanou instanci poskytovatele `SqlMembershipProvider` definovanou v souboru `machine.config` (`AspNetSqlMembershipProvider`, což je hodnota atributu `name` v deklaraci poskytovatele). Zvolíte-li tento přístup, musíte vymazat předchozí definovanou instanci `AspNetSqlMembershipProvider`. Je zapotřebí předefinovat `AspNetSqlMembershipProvider` pomocí uzlu `<clear />` v sekci `<providers>`. Pokud byste to neučinili, objevila by se chyba říkající, že daný název poskytovatele je již definován.

Jakmile vymažete předchozí instanci `AspNetSqlMembershipProvider`, předefinujete tohoto poskytovatele elementem `<add>`. V případě výpisu 13.2 je zřejmé předefinování požadavků na heslo použitím nových hodnot atributů `minRequiredPasswordLength` a `minRequiredNonalphanumericCharacters` (zvýrazněno).

Jiný přístup k definování vlastní instance `SqlMembershipProvider` spočívá v tom, že poskytovateli definovanému v elementu `<add>` přiřadíte hodnotu atributu `name`. V takovém případě musíte určit tuto novou pojmenovanou instanci za výchozího poskytovatele systému členství atributem `defaultProvider`. Tento přístup ukazuje výpis 13.3.

Výpis 13.3: Definování vlastní pojmenované instance SqlMembershipProvider

```
<membership defaultProvider="MyVeryOwnAspNetSqlMembershipProvider">
  <providers>
    <add name="MyVeryOwnAspNetSqlMembershipProvider"
```

```
      type="System.Web.Security.SqlMembershipProvider,
        System.Web, Version=2.0.0.0, Culture=neutral,
        PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="LocalSqlServer"
      enablePasswordRetrieval="false"
      enablePasswordReset="true"
      requiresQuestionAndAnswer="true"
      requiresUniqueEmail="false"
      passwordFormat="Hashed"
      maxInvalidPasswordAttempts="5"
      minRequiredPasswordLength="4"
      minRequiredNonalphanumericCharacters="0"
      passwordAttemptWindow="10" />
  </providers>
</membership>
```

Nyní se definice instance poskytovatele `SqlMembershipProvider` v souboru `machine.config` (pod názvem `AspNetSqlMembershipProvider`) vůbec nemění. Místo toho se tu v souboru `web.config` definuje úplně nová pojmenovaná instance (`MyVeryOwnAspNetSqlMembershipProvider`).

Silnější struktury hesel díky SqlMembershipProvider

Nyní si ukážeme, jak naopak strukturu hesel trochu zkomplikovat. Toho lze pochopitelně dosáhnout několika způsoby. Jedním z nich je použít stejné atributy `minRequiredPasswordLength` a `minRequiredNonalphanumericCharacters` (jak již byly ukázány) a požadovat větší délku hesla (což většinou znamená vyšší bezpečnost) i zahrnutí určitého počtu jiných než alfanumerických znaků (což také zvyšuje bezpečnost hesla).

Další možností je použít atribut `passwordStrengthRegularExpression`. Pokud vám atributy `minRequiredPasswordLength` a `minRequiredNonalphanumericCharacters` nemohou zajistit vyžadovanou strukturu hesla, je další dobrou alternativou právě atribut `passwordStrengthRegularExpression`.

Využití zmíněného atributu si ukážeme na příkladu, kdy má uživatel zadat jako heslo číslo svého sociálního pojištění v USA. Pak nadefinujeme poskytovatele způsobem zachyceným ve výpisu 13.4.

Výpis 13.4: Instance poskytovatele v souboru web.config, která mění strukturu hesla

```
<configuration>

  <system.web>

    <authentication mode="Forms" />

    <membership>
      <providers>
        <clear />
        <add name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider,
            System.Web, Version=2.0.0.0, Culture=neutral,
            PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="LocalSqlServer"
```

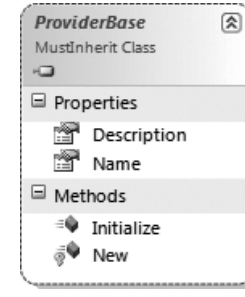
```
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="true"
          requiresUniqueEmail="false"
          passwordFormat="Hashed"
          maxInvalidPasswordAttempts="5"
          passwordAttemptWindow="10"
          passwordStrengthRegularExpression="\d{3}-\d{2}-\d{4}" />
      </providers>
    </membership>

  </system.web>

</configuration>
```

Místo atributů `minRequiredPasswordLength` a `minRequiredNonalphanumericCharacters` se aplikuje atribut `passwordStrengthRegularExpression`, jemuž je přiřazena hodnota `\d{3}-\d{2}-\d{4}`. Uvedený regulární výraz znamená, že heslo musí obsahovat tři číslice, pak pomlčku neboli rozdělovník, následně dvě číslice, další rozdělovník a nakonec čtyři číslice.

Měli byste již chápat, že existuje mnoho možností, jak upravit chování poskytovatelů dostupných po instalaci rámce .NET Framework 3.5. Řadu vestavěných poskytovatelů si můžete přizpůsobit pomocí programování atributů. Ukázka poskytovatele `SqlMembershipProvider` to vše představila, ovšem stejně snadno lze zadat podobné změny ostatním poskytovatelům.

Seznámení s ProviderBase

Všichni poskytovatelé jsou nějakým způsobem odvozeni ze třídy `ProviderBase`, která se nachází v oboru názvů `System.Configuration.Provider`. `ProviderBase` je abstraktní třída používaná k definování základní šablony pro dědící poskytovatele. Když se podíváte na abstraktní třídu `ProviderBase`, všimněte si, že toho mnoho neobsahuje (viz obrázek 13.2).



Obrázek 13.2

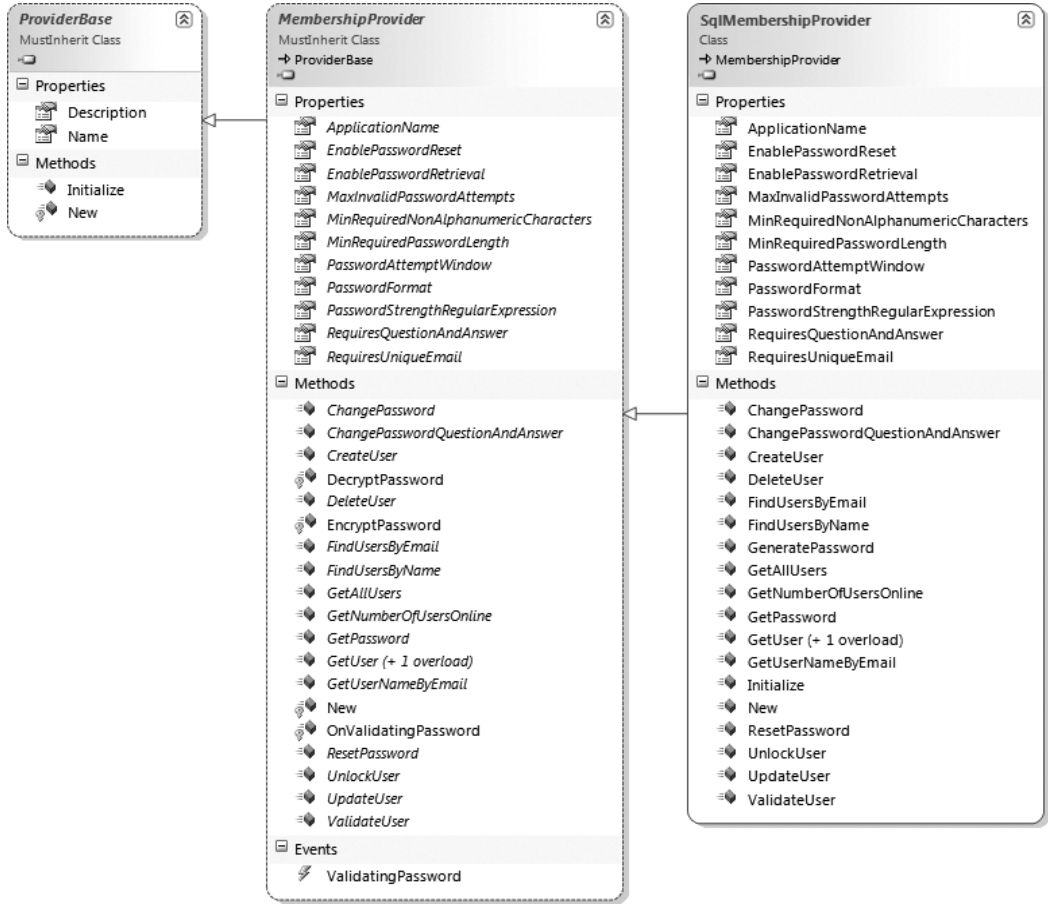
Jak bylo řečeno, tato třída mnoho nenabízí. Vlastně je to jen kořenová třída pro poskytovatele, aby se mohli inicializovat.

Vlastnost `Name` se používá k zadání srozumitelného názvu, kupříkladu `AspNetSqlRoleProvider`. Vlastnost `Description` umožňuje zadat textový popis poskytovatele, který mohou následně využívat nástroje pro správu. Hlavní položkou ve třídě `ProviderBase` je metoda `Initialize()`. Zde máme konstruktor metody `Initialize()`:

```
public virtual void Initialize(string name,
  System.Collections.Specialized.NameValueCollection config);
```

Všimněte si dvou parametrů metody `Initialize()`. První z nich je prostě hodnotou přiřazenou atributu `name` v deklaraci poskytovatele v konfiguračním souboru. Parametr `config` je typu `NameValueCollection`, což je kolekce dvojic název/hodnota. Tyto dvojice název/hodnota jsou položkami rovněž definovanými v deklaraci daného poskytovatele v konfiguračním souboru jako různé atributy a jim přiřazené hodnoty.

Když se podíváte na poskytovatele obsažené ve výchozí instalaci ASP.NET 3.5, zjistíte, že každý z nich definuje třídu, z níž můžete dědit a jež implementuje abstraktní třídu `ProviderBase`. Kupříkladu v modelu zajišťujícím systém členství najdete instanci `MembershipProvider`, která se dědí v konečné deklaraci `SqlMembershipProvider`. Třída `MembershipProvider` však implementuje samotnou `ProviderBase`. Tento model najdete na obrázku 13.3.



Obrázek 13.3

Všimněte si, že každý z různých systémů má konkrétní implementaci bázového poskytovatele, s níž můžete pracovat.

Skutečně nemůže existovat jediný poskytovatel naplňující potřeby všech dostupných systémů. Z obrázku 13.3 je zřejmé, že instance `MembershipProvider` vystavuje určitou velmi specifickou funkčnost vyžadovanou systémem členství ASP.NET. Vystavované metody určitě nepotřebuje systém správy rolí ani podpora modulů WebPart.

Díky různým existujícím základním implementacím máte při vytváření vlastních přizpůsobení systému členství ASP.NET několik možností. Zaprvé, můžete prostě vytvořit svého vlastního poskytovatele přímo implementací třídy `ProviderBase`, takže všechno budete zajišťovat sami. Tento přístup ovšem nedoporučuji, protože máte k dispozici abstraktní třídy využitelné v různých systémech. Jak jsem tedy zmínil, stačí jen implementovat instanci `MembershipProvider` (což je lepší možnost) a pracovat s jí zajišťovaným modelem. Jestliže využíváte nějakou funkčnost SQL Serveru a jen chcete změnit základní chování odpovídajícího poskytovatele, můžete dědit z `SqlMembershipProvider` a upravit si chování této třídy. Nyní se podíváme na příklady různých možností rozšíření modelu poskytovatelů.

Sestavení vlastních poskytovatelů

Nyní se podíváme blíže na proces sestavení vlastního poskytovatele využitelného dále v nějaké aplikaci ASP.NET 3.5. Vlastně není vůbec obtížné sestavovat poskytovatele (jak za chvilku uvidíte); dokonce tak můžete učinit přímo v projektu ASP.NET 3.5. Náš příklad zachycuje sestavení poskytovatele členství pracujícího se souborem XML. V případě menšího webového sídla se může jednat o dobře využitelný princip. U větších webových sídel a webových aplikací je při správě uživatelů lepší pracovat s nějakou formou databáze, a nikoli se souborem XML.

Při sestavování svého vlastního poskytovatele členství máte několik možností. Požadované funkčnosti lze dosáhnout odvozením ze třídy `SqlMembershipProvider` nebo `MembershipProvider`. Ze `SqlMembershipProvider` budete odvozovat, pouze potřebujete-li rozšířit nebo změnit chování systému členství ve vztahu k SQL. Protože naším cílem je sestavit poskytovatele členství, který bude pouze ke čtení a bude využívat soubor XML, odvozování z výše uvedené třídy není vhodné. V našem případě je mnohem lepší vycházet ze třídy `MembershipProvider`.

Vytvoření aplikace CustomProviders

V tomto příkladu vytvořte ve svém oblíbeném jazyku nový projekt webového sídla nazvaný CustomProviders. Sestavíme v něm nového poskytovatele členství, a to přímo v dané webové aplikaci. Další možností je sestavit poskytovatele v projektu knihovny tříd (Class Library) a následně se odkazovat na vygenerovanou knihovnu DLL ve webovém projektu. V obou případech dosáhnete stejného výsledku.

Protože budeme sestavovat poskytovatele přímo v projektu webového sídla, vytvoříme v aplikaci složku `App_Code`. Na toto místo budeme vkládat dále vytvářený soubor třídy, který bude v našem případě vlastním poskytovatelem.

Jakmile máme složku `App_Code`, vytvoříme v ní novou třídu, kterou nazveme `XmlMembershipProvider.vb` nebo `XmlMembershipProvider.cs` podle vybraného jazyka. Když třídu `XmlMembershipProvider` vytvoříme, můžeme dědit z `MembershipProvider`. Abychom toho dosáhli a zároveň viděli, které metody a vlastnosti je zapotřebí překrýt, použijeme Visual Studio 2008 k sestavení kostry naší třídy. Celým procesem projdeme v jednotlivých krocích a kódem odpovídajícím výpisu 13.5.

Výpis 13.5: Začátek třídy XmlMembershipProvider

VB

```
Imports Microsoft.VisualBasic
Imports System.Xml
```

```
Imports System.Configuration.Provider
Imports System.Web.Hosting
Imports System.Collections
Imports System.Collections.Generic

Public Class XmlMembershipProvider
  Inherits MembershipProvider

End Class
```

C#

```
using System;
using System.Web.Hosting;
using System.Web.Security;
using System.Xml;
using System.Collections.Generic;

/// <summary>
/// Shrnující popis pro třídu XmlMembershipProvider
/// </summary>
public class XmlMembershipProvider: MembershipProvider
{
  public XmlMembershipProvider()
  {
    //
    // Udělat: Sem přidat logický konstruktor.
    //
  }
}
```

V základní třídě `XmlMembershipProvider` provedeme jen několik změn. Především je patrné importování dodatečných oborů názvů do souboru. To je proto, abychom mohli později využít podporu XML, generiku a další prvky rámce .NET. Také si všimněte, že tato nová třída `XmlMembershipProvider` dědí od `MembershipProvider`.

Konstrukce potřebné kostry třídy

Má-li Visual Studio 2008 sestavit třídu s odpovídajícími metodami a vlastnostmi, proveďte následující kroky (v závislosti na používaném jazyku): Pracujete-li ve Visual Basicu, pak stačí jen stisknout klávesu Enter. V C# nejprve umístěte kurzor do instance `MembershipProvider` v okně dokumentu a pak zadejte příkaz Edit → IntelliSense → Implement Abstract Class nabídky Visual Studia. Po vykonání jedné z uvedených operací spatříte v okně dokumentu Visual Studia úplnou kostru nové třídy. Výpis 13.6 ukazuje kód vygenerovaný pro třídy `XmlMembershipProvider` ve Visual Basicu.

Výpis 13.6: Kód vygenerovaný pro třídu XmlMembershipProvider Visual Studiem

VB (pouze)

```
Imports Microsoft.VisualBasic
Imports System.Xml
Imports System.Configuration.Provider
Imports System.Web.Hosting
```

```
Imports System.Collections
Imports System.Collections.Generic

Public Class XmlMembershipProvider
  Inherits MembershipProvider

  Public Overrides Property ApplicationName() As String
    Get

    End Get
    Set(ByVal value As String)

    End Set
  End Property

  Public Overrides Function ChangePassword(ByVal username As String, _
    ByVal oldPassword As String, ByVal newPassword As String) As Boolean

  End Function

  Public Overrides Function ChangePasswordQuestionAndAnswer(ByVal username _
    As String, ByVal password As String, ByVal newPasswordQuestion As String, _
    ByVal newPasswordAnswer As String) As Boolean

  End Function

  Public Overrides Function CreateUser(ByVal username As String, _
    ByVal password As String, ByVal email As String, _
    ByVal passwordQuestion As String, ByVal passwordAnswer As String, _
    ByVal isApproved As Boolean, ByVal providerUserKey As Object, _
    ByRef status As System.Web.Security.MembershipCreateStatus) As _
    System.Web.Security.MembershipUser

  End Function

  Public Overrides Function DeleteUser(ByVal username As String, _
    ByVal deleteAllRelatedData As Boolean) As Boolean

  End Function

  Public Overrides ReadOnly Property EnablePasswordReset() As Boolean
    Get

    End Get
  End Property

  Public Overrides ReadOnly Property EnablePasswordRetrieval() As Boolean
    Get

    End Get
  End Property
```

```
Public Overrides Function FindUsersByEmail(ByVal emailToMatch As String, _
  ByVal pageIndex As Integer, ByVal pageSize As Integer, _
  ByRef totalRecords As Integer) As _
  System.Web.Security.MembershipUserCollection

End Function

Public Overrides Function FindUsersByName(ByVal usernameToMatch As String, _
  ByVal pageIndex As Integer, ByVal pageSize As Integer, _
  ByRef totalRecords As Integer) As _
  System.Web.Security.MembershipUserCollection

End Function

Public Overrides Function GetAllUsers(ByVal pageIndex As Integer, _
  ByVal pageSize As Integer, ByRef totalRecords As Integer) As _
  System.Web.Security.MembershipUserCollection

End Function

Public Overrides Function GetNumberOfUsersOnline() As Integer

End Function

Public Overrides Function GetPassword(ByVal username As String, _
  ByVal answer As String) As String

End Function

Public Overloads Overrides Function GetUser(ByVal providerUserKey As Object, _
  ByVal userIsOnline As Boolean) As System.Web.Security.MembershipUser

End Function

Public Overloads Overrides Function GetUser(ByVal username As String, _
  ByVal userIsOnline As Boolean) As System.Web.Security.MembershipUser

End Function

Public Overrides Function GetUserNameByEmail(ByVal email As String) As String

End Function

Public Overrides ReadOnly Property MaxInvalidPasswordAttempts() As Integer
  Get

  End Get
End Property

Public Overrides ReadOnly Property MinRequiredNonAlphanumericCharacters() _
  As Integer
  Get
```

```
    End Get
  End Property

  Public Overrides ReadOnly Property MinRequiredPasswordLength() As Integer
    Get

    End Get
  End Property

  Public Overrides ReadOnly Property PasswordAttemptWindow() As Integer
    Get

    End Get
  End Property

  Public Overrides ReadOnly Property PasswordFormat() As _
    System.Web.Security.MembershipPasswordFormat
    Get

    End Get
  End Property

  Public Overrides ReadOnly Property PasswordStrengthRegularExpression() As _
    String
    Get

    End Get
  End Property

  Public Overrides ReadOnly Property RequiresQuestionAndAnswer() As Boolean
    Get

    End Get
  End Property

  Public Overrides ReadOnly Property RequiresUniqueEmail() As Boolean
    Get

    End Get
  End Property

  Public Overrides Function ResetPassword(ByVal username As String, _
    ByVal answer As String) As String

  End Function

  Public Overrides Function UnlockUser(ByVal userName As String) As Boolean

  End Function

  Public Overrides Sub UpdateUser(ByVal user As _
```

```
    System.Web.Security.MembershipUser)

  End Sub

  Public Overrides Function ValidateUser(ByVal username As String, _
    ByVal password As String) As Boolean

  End Function
End Class
```

Páni, to je spousta kódu! Kostru už sice máme, ovšem nyní je zapotřebí vytvořit některé z položek, které bude poskytovatel nabízený Visual Studiem skutečně používat – začneme souborem XML, jenž bude obsahovat všechny uživatele, kteří mají k aplikaci přístup.

Vytvoření datového skladu uživatelů ve formátu XML

Jelikož se jedná o poskytovatele členství využívajícího XML, je naším cílem načítat údaje o uživatelích ze souboru XML a nikoli z databáze, jako je SQL Server. Proto musíme definovat strukturu souboru XML, kterou může poskytovatel využívat. Strukturu našeho příkladu najdete ve výpisu 13.7.

Výpis 13.7: Soubor XML použitý k ukládání uživatelských jmen a hesel

```
<?xml version="1.0" encoding="utf-8" ?>
<Users>
  <User>
    <Username>BillEvjen</Username>
    <Password>Bubbles</Password>
    <Email>evjen@yahoo.com</Email>
    <DateCreated>10.11.2008</DateCreated>
  </User>
  <User>
    <Username>ScottHanselman</Username>
    <Password>YabbaDabbaDo</Password>
    <Email>123@msn.com</Email>
    <DateCreated>20.10.2008</DateCreated>
  </User>
  <User>
    <Username>DevinRader</Username>
    <Password>BamBam</Password>
    <Email>456@msn.com</Email>
    <DateCreated>23.9.2008</DateCreated>
  </User>
</Users>
```

Uvedený soubor XML obsahuje jen tři instance uživatelů, které všechny zahrnují uživatelské jméno, heslo, adresu elektronické pošty a datum vytvoření daného uživatele. Jelikož se jedná o datový soubor, umístíme jej do složky `App_Data` aplikace ASP.NET. Soubor lze pojmenovat libovolně, my ovšem použijeme název `UserDatabase.xml`.

Dále se podíváme, jak převzít uložené hodnoty ze souboru XML při ověřování uživatelů.

Definování instance poskytovatele v souboru web.config

Jak jste viděli v předchozí kapitole o poskytovatelích, definujete je samotné i jejich chování v konfiguračním souboru (kupříkladu `machine.config` nebo `web.config`). Jelikož sestavujeme poskytovatele pro jedinou aplikaci, budeme jej definovat v odpovídajícím souboru `web.config`.

Výchozím poskytovatelem je `SqlMembershipProvider`, který je zadán v souboru `machine.config` na serveru. Naším úkolem je překrýt toto nastavení a vytvořit nového výchozího poskytovatele. Deklarace poskytovatele členství XML by měla v souboru `web.config` vypadat tak, jak to ukazuje výpis 13.8.

Výpis 13.8: Definování poskytovatele XmlMembershipProvider v souboru web.config

```
<configuration>
  <system.web>

    <authentication mode="Forms"/>

    <membership defaultProvider="XmlFileProvider">
      <providers>
        <add name="XmlFileProvider" type="XmlMembershipProvider"
          xmlUserDatabaseFile="~/App_Data/UserDatabase.xml"/>
      </providers>
    </membership>

  </system.web>
</configuration>
```

Z výpisu je vidět, že za výchozího poskytovatele je zadán `XmlFileProvider`. Protože tento název poskytovatele nebude v žádných nadřazených konfiguračních souborech nalezen, musíme jej definovat v souboru `web.config`.

S využitím atributu `defaultProvider` je možné definovat název poskytovatele, kterého má využívat systém členství. V našem případě se jedná o `XmlFileProvider`. Dále se definuje instance třídy `XmlFileProvider` pomocí elementu `<add>` v sekci `<providers>`. Element `<add>` poskytovatele pojmenovává – `XmlFileProvider`. Zároveň ukazuje na třídu (neboli typ) poskytovatele. Zde se jedná o vytvořenou kostru třídy – `XmlMembershipProvider`. To jsou dva nejdůležitější atributy.

Kromě výše uvedených můžete vytvořit v deklaraci poskytovatele libovolné další atributy. Ať už vytvoříte jakýkoli typ poskytovatele, musíte v něm odpovídající atributy zpracovávat a fungovat na základě v nich zadaných hodnot. V případě jednoduchého poskytovatele `XmlMembershipProvider` využijeme jen jeden vlastní atribut – `xmlUserDatabaseFile`. Tento atribut ukazuje na místo souboru XML zahrnujícího databázi uživatelů. Bude se jednat o volitelný atribut; pokud nebude hodnota atributu `xmlUserDatabaseFile` zadána, využije se výchozí hodnota. Ve výpisu 13.8 si ovšem všimněte, že hodnotu odpovídající využívanému souboru XML skutečně zadáváme. Hodnotou `xmlUserDatabaseFile` je jen název souboru, nic jiného.

V příkladu není ukázán jiný atribut, který je ovšem podporován díky třídě `XmlMembershipProvider`. Atribut `applicationName` ukazuje na aplikaci, které má příslušné instanci `XmlMembershipProvider` sloužit. Výchozí hodnotu, kterou můžete vložit také do deklarace poskytovatele v konfiguračním souboru, najdete zde:

```
applicationName="/"
```

Neimplementování metod a vlastností třídy MembershipProvider

Nyní obraťme svou pozornost ke třídě `XmlMembershipProvider`. Dalším krokem je implementovat všechny metody a vlastnosti, jež daný poskytovatel potřebuje. Není nutné skutečně používat všechny metody obsažené v kostře; stačí jen sestavit ty metody, které jsou pro vás podstatné. Pokud kupříkladu neumožňujete programovou změnu hesel (a nepodporujete tedy ani ovládací prvky, jež takový programový přístup využívají), pak nebudete chtít takovou akci vůbec spouštět nebo dokonce budete vyvolávat chybu, když se někdo o implementování této metody pokusí. To zachycuje výpis 13.9.

Výpis 13.9: Neimplementování jedné z dostupných metod vyvoláním výjimky

VB

```
Public Overrides Function ChangePassword(ByVal username As String, _
  ByVal oldPassword As String, ByVal newPassword As String) As Boolean
  Throw New NotSupportedException()
End Function
```

C#

```
public override bool ChangePassword(string username,
string oldPassword, string newPassword)
{
  throw new NotSupportedException();
}
```

V tomto případě se při volání metody `ChangePassword()` vyvolá výjimka `NotSupportedException`. Nechcete-li vyvolávat skutečnou výjimku, můžete jen vrátit hodnotu `false` a neprovádět nic jiného, jak to ukazuje výpis 13.10. (Ovšem tento přístup může být nepříjemný pro další vývojáře, kteří se snaží metodu implementovat a neznají její vnitřní logiku.)

Výpis 13.10: Neimplementování jedné z dostupných metod vrácením hodnoty false

VB

```
Public Overrides Function ChangePassword(ByVal username As String, _
  ByVal oldPassword As String, ByVal newPassword As String) As Boolean
  Return False
End Function
```

C#

```
public override bool ChangePassword(string username,
  string oldPassword, string newPassword)
{
  return false;
}
```

Tato kapitola nepopisuje všechny možné akce proveditelné s poskytovatelem `XmlMembershipProvider`, a proto bude vhodné, když si projdete dostupné metody a vlastnosti odvozené instance `MembershipProvider` a provedete potřebné změny u všech položek, jež nebudete používat.

Implementování metod a vlastností třídy MembershipProvider

Nastal čas implementovat některé z metod a vlastností třídy `MembershipProvider`, aby náš poskytovatel `XmlMembershipProvider` začal fungovat. Nejprve jsou v deklaraci nějaké soukromé proměnné využitelné více metodami v celé třídě. Ve výpisu 13.11 máme deklarace zmíněných proměnných.

Výpis 13.11: Deklarování některých soukromých proměnných ve třídě XmlMembershipProvider

VB

```
Public Class XmlMembershipProvider
  Inherits MembershipProvider

  Private _AppName As String
  Private _MyUsers As Dictionary(Of String, MembershipUser)
  Private _FileName As String

  ' Kód odstraněn kvůli přehlednosti
End Class
```

C#

```
public class XmlMembershipProvider: MembershipProvider
{
  private string _AppName;
  private Dictionary<string, MembershipUser> _MyUsers;
  private string _FileName;

  // Kód odstraněn kvůli přehlednosti
}
```

Deklarované proměnné jsou položkami, které potřebuje více metod v dané třídě. Proměnná `_AppName` definuje aplikaci používající poskytovatele členství XML. Ve všech případech se jedná o lokální aplikaci. Také je dobré umístit všechny členy ze souboru XML do nějaké kolekce. Náš příklad využívá generický typ slovníku nazvaný `_MyUsers`. Příklad nakonec ukazuje na používaný soubor proměnnou `_FileName`.

Vlastnost ApplicationName

Jakmile máme soukromé proměnné, můžeme definovat vlastnost `ApplicationName`. K tomu využijeme první soukromou proměnnou `_AppName`. Definici vlastnosti `ApplicationName` najdete ve výpisu 13.12.

Výpis 13.12: Definování vlastnosti ApplicationName

VB

```
Public Overrides Property ApplicationName() As String
  Get
    Return _AppName
  End Get
  Set(ByVal value As String)
    _AppName = value
```

```
  End Set
End Property
```

C#

```
public override string ApplicationName
{
  get
  {
    return _AppName;
  }
  set
  {
    _AppName = value;
  }
}
```

Protože je vlastnost `ApplicationName` definována a připravena, můžeme převzít hodnoty z deklarace poskytovatele (`XmlFileProvider`) v souboru `web.config`.

Rozšíření metody Initialize()

Nyní rozšíříme metodu `Initialize()`, aby načítala vlastní atribut a jemu přiřazenou hodnotu podle deklarace poskytovatele v souboru `web.config`. Projděte si kostrou třídy `XmlMembershipProvider` a všimněte si, že na seznamu dostupných položek metoda `Initialize()` není.

Metoda `Initialize()` se volá při první inicializaci daného poskytovatele. Není nutné tuto metodu překrývat, a proto se neobjeví v deklaraci kostry třídy. Chcete-li vložit vlastní metodu `Initialize()` do třídy `XmlMembershipProvider`, pak do ní jen zadejte `Public Overrides` (v případě Visual Basicu) nebo `public override` (v C#). Pak vám funkce IntelliSense nabídne metodu `Initialize()` (viz obrázek 13.4).

Vložení metody `Initialize()` do třídy tímto způsobem je velmi snadné. Vyberte metodu `Initialize()` v seznamu IntelliSense a stiskněte klávesu Enter. Tím získáte základní konstrukci metody v kódu, jak to zachycuje výpis 13.13.

Výpis 13.13: Začátky metody Initialize

VB

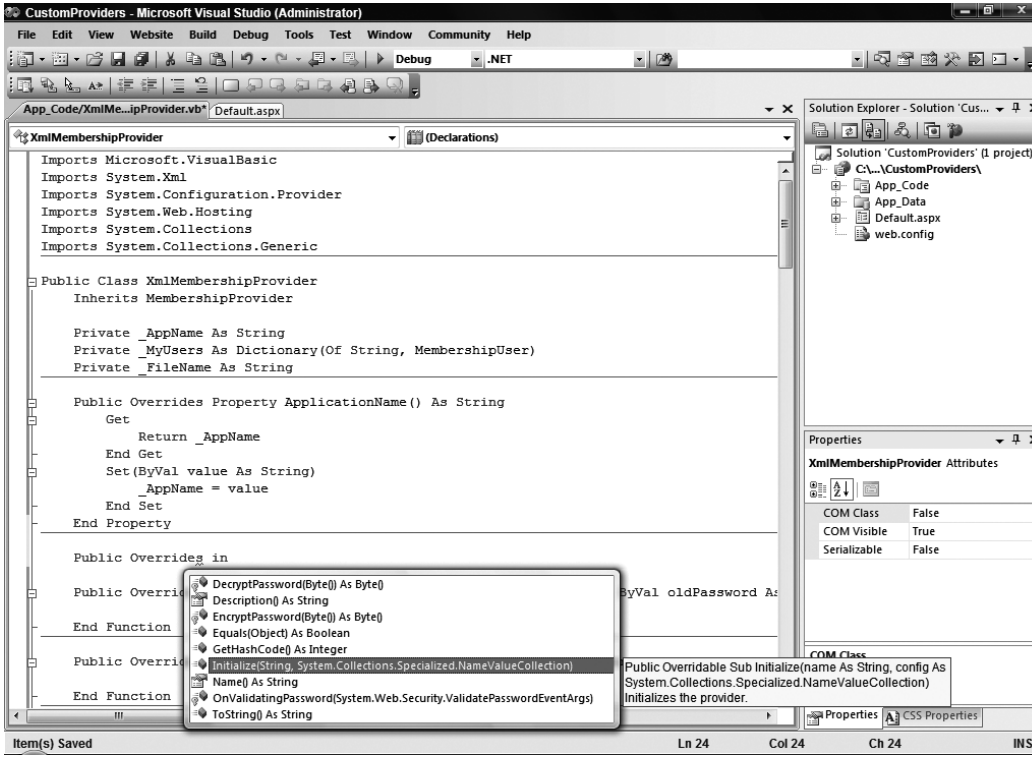
```
Public Overrides Sub Initialize(ByVal name As String, _
  ByVal config As System.Collections.Specialized.NameValueCollection)
  MyBase.Initialize(name, config)
End Sub
```

C#

```
public override void Initialize(string name,
  System.Collections.Specialized.NameValueCollection config)
{
  base.Initialize(name, config);
}
```

Metoda `Initialize()` přebírá dva parametry. Prvním je název parametru a druhým je kolekce názvů/hodnot z deklarace poskytovatele v souboru `web.config`. Ta zahrnuje všechny atributy a jejich

hodnoty, jako například atribut `xmlUserDatabaseFile` a hodnotu názvu souboru XML, jenž obsahuje údaje o uživatelích. Pomocí konfigurace můžete získat přístup k takto definovaným hodnotám.



Obrázek 13.4

V případě instance `XmlFileProvider` využíváme atribut `applicationName` a atribut `xmlUserDatabaseFile`. Vše potřebné ukazuje výpis 13.14.

Výpis 13.14: Rozšíření metody Initialize()

VB

```
Public Overrides Sub Initialize(ByVal name As String, _
  ByVal config As System.Collections.Specialized.NameValueCollection)

  MyBase.Initialize(name, config)
  _AppName = config("applicationName")

  If (String.IsNullOrEmpty(_AppName)) Then
    _AppName = "/"
  End If

  _FileName = config("xmlUserDatabaseFile")
```

```
  If (String.IsNullOrEmpty(_FileName)) Then
    _FileName = "~/App_Data/Users.xml"
  End If
End Sub
```

C#

```
public override void Initialize(string name,
  System.Collections.Specialized.NameValueCollection config)
{
  base.Initialize(name, config);

  _AppName = config["applicationName"];

  if (String.IsNullOrEmpty(_AppName))
  {
    _AppName = "/";
  }

  _FileName = config["xmlUserDatabaseFile"];

  if (String.IsNullOrEmpty(_FileName))
  {
    _FileName = "~/App_Data/Users.xml";
  }
}
```

Kromě zajištění inicializace s využitím `MyBase.Initialize()` přebíráme hodnoty atributů `applicationName` i `xmlUserDatabaseFile` přes konfiguraci. V každém případě je nejprve zapotřebí prověřit, zda není hodnota `null` nebo prázdná. K přiřazení výchozích hodnot, když atribut v deklaraci poskytovatele v souboru `web.config` chybí, se používá metoda `String.IsNullOrEmpty()`. V případě instance `XmlFileProvider` je to právě náš případ. Atribut `applicationName` není v deklaraci `XmlFileProvider` obsažen, a tak se mu přiřadí výchozí hodnota `/`.

U atributu `xmlUserDatabaseFile` je zadána hodnota. Pokud by se ovšem v souboru `web.config` nenacházela, hledal by poskytovatel soubor XML nazvaný `Users.xml` ve složce `App_Data`.

Ověření uživatelů

Jednou z důležitějších funkcí poskytovatele členství je ověřování uživatelů (jejich autentikace). Ověřování uživatelů zajišťuje serverový ovládací prvek Login ASP.NET. Ten zase využívá metodu `Membership.ValidateUser()`, jež sama nakonec volá metodu `ValidateUser()` ve třídě `XmlMembershipProvider`.

Jelikož už máme metodu `Initialize()` a soukromé proměnné, můžeme začít vybavovat poskytovatele využitelnou funkčností. Implementaci metody `ValidateUser()` najdete ve výpisu 13.15.

Výpis 13.15: Implementování metody ValidateUser()

VB

```
Public Overrides Function ValidateUser(ByVal username As String, _
  ByVal password As String) As Boolean
```

```
  If (String.IsNullOrEmpty(username) Or String.IsNullOrEmpty(password)) Then
    Return False
  End If

  Try
    ReadUserFile()
    Dim mu As MembershipUser

    If (_MyUsers.TryGetValue(username.ToLower(), mu)) Then
      If (mu.Comment = password) Then
        Return True
      End If
    End If

    Return False

  Catch ex As Exception
    Throw New Exception(ex.Message.ToString())
  End Try

End Function
```

C#

```
public override bool ValidateUser(string username, string password)
{
  if (String.IsNullOrEmpty(username) || String.IsNullOrEmpty(password))
  {
    return false;
  }

  try
  {
    ReadUserFile();
    MembershipUser mu;

    if (_MyUsers.TryGetValue(username.ToLower(), out mu))
    {
      if (mu.Comment == password)
      {
        return true;
      }
    }
    return false;
  }
  catch (Exception ex)
  {
    throw new Exception(ex.Message.ToString());
  }
}
```

Z kódu metody `ValidateUser()` je patrné, že přebírá dva parametry, konkrétně uživatelské jméno a heslo příslušného uživatele (oba typu `String`). Návratová hodnota metody `ValidateUser()` je logická – jen `True` nebo `False` informující o úspěchu či selhání procesu ověření.

Jednou z prvních operací vykonanou v metodě `ValidateUser()` je kontrola toho, zda ve volání nechybí uživatelské jméno nebo heslo. Pokud některý z těchto dvou řetězců chybí, je vrácena hodnota `False`.

Následující kontrola `Try Catch` prověřuje, zda je odpovídající jméno a heslo obsaženo v souboru XML. Proces převzetí údajů o uživatelích ze souboru XML a do proměnné `MyUsers` zajišťuje metoda `ReadUserFile()`. Tuto metodu si za chvilku popíšeme, důležité je ovšem to, že proměnná `_MyUsers` je instancí generické třídy `Dictionary`. Klíčem je řetězec malých znaků představující uživatelské jméno, zatímco hodnota je typu `MembershipUser`, který poskytuje systém členství.

Jakmile je objekt `_MyUsers` naplněn všemi uživateli ze souboru XML, vytvoří se instance `MembershipUser`. Tento objekt je výstupem operace `TryGetValue`. Objekt `MembershipUser` neobsahuje heslo uživatele, a proto musí heslo uživatele zjistit metoda `ReadUserFile()`, která z něj učiní hodnotu vlastnosti `Comment` třídy `MembershipUser`. Je-li ve slovníkové kolekci nalezeno uživatelské jméno, pak se heslo této konkrétní instance `MembershipUser` porovná s hodnotou vlastnosti `Comment`. Návratovou hodnotou metody `ValidateUser()` je `True`, pokud si obě hodnoty odpovídají.

Jak vidíte, popisovaná metoda zásadně závisí na výsledcích pocházejících z metody `ReadUserFile()`, kterou si nyní přiblížíme.

Sestavení metody ReadUserFile()

Metoda `ReadUserFile()` načítá obsah souboru XML, jenž obsahuje všechny uživatele aplikace. Tato metoda je vlastní a funguje mimo rámec metody `ValidateUser()`. To znamená, že ji lze opakovaně využívat i v jiných metodách, které můžete implementovat (kupříkladu `GetAllUsers()`). Jediným úkolem metody `ReadUserFile()` je načíst obsah příslušného souboru XML a umístit všechny uživatele do proměnné `_MyUsers`, jak to ukazuje výpis 13.16.

Výpis 13.16: Metoda ReadUserFile() přebírající všechny uživatele aplikace

VB

```
Private Sub ReadUserFile()
  If (_MyUsers Is Nothing) Then
    SyncLock (Me)
      _MyUsers = New Dictionary(Of String, MembershipUser)()
      Dim xd As XmlDocument = New XmlDocument()
      xd.Load(HostingEnvironment.MapPath(_FileName))
      Dim xnl As XmlNodeList = xd.GetElementsByTagName("User")

      For Each node As XmlNode In xnl
        Dim mu As MembershipUser = New MembershipUser(Name, _
          node("Username").InnerText, _
          Nothing, _
          node("Email").InnerText, _
          String.Empty, _
          node("Password").InnerText, _
          True, _
```

```
          False, _
          DateTime.Parse(node("DateCreated").InnerText), _
          DateTime.Now, _
          DateTime.Now, _
          DateTime.Now, _
          DateTime.Now)

        _MyUsers.Add(mu.UserName.ToLower(), mu)
      Next
    End SyncLock
  End If
End Sub
```

C#

```
private void ReadUserFile()
{
  if (_MyUsers == null)
  {
    lock (this)
    {
      _MyUsers = new Dictionary<string, MembershipUser>();
      XmlDocument xd = new XmlDocument();
      xd.Load(HostingEnvironment.MapPath(_FileName));
      XmlNodeList xnl = xd.GetElementsByTagName("User");

      foreach (XmlNode node in xnl)
      {
        MembershipUser mu = new MembershipUser(Name,
          node["Username"].InnerText,
          null,
          node["Email"].InnerText,
          String.Empty,
          node["Password"].InnerText,
          true,
          false,
          DateTime.Parse(node["DateCreated"].InnerText),
          DateTime.Now,
          DateTime.Now,
          DateTime.Now,
          DateTime.Now);

        _MyUsers.Add(mu.UserName.ToLower(), mu);
      }
    }
  }
}
```

První akcí metody `ReadUserFile()` je uzamknout v běžícím vlákně činnost, která bude následovat. Jedná se o jedinečnou funkčnost ASP.NET. Když píšete své vlastní poskytovatele, musíte vždy pracovat ve vláknově zabezpečeném kódu. Většina položek ASP.NET, jako je HttpModule nebo http-Handler (jak je popisuje kapitola 27), nemusí být vláknově zabezpečená. Takové položky mohou

vykonávat více požadavků ve více vláknech, přičemž každé vlákno požadující něco od HttpModule nebo HttpHandler vidí jeho jedinečnou instanci.

Na rozdíl od HttpHandler vytváří a využívá aplikace ASP.NET jediného poskytovatele. Pokud směřuje na vaši aplikaci více požadavků, tak se všechna taková vlákna pokoušejí o získání přístupu k této jediné instanci poskytovatele obsažené v aplikaci. Jelikož může najednou přicházet k instanci poskytovatele více požadavků, je zapotřebí vytvořit poskytovatele vláknově zabezpečeného. Toho dosáhnete aplikováním operace uzamčení (lock) na vykonávání takových činností, jako jsou souborové operace I/O. Právě proto se používá příkaz `SyncLock` (ve Visual Basicu) a `lock` (v C#) v metodě `ReadUserFile()`.

Výhodou tohoto uspořádání je ovšem to, že v aplikaci běží jediná instance poskytovatele. Jakmile je objekt `_MyUsers` naplněn obsahem souboru XML, není zapotřebí znovu jej naplňovat. Instance poskytovatele prostě nezmizí po odeslání odpovědi žadateli. Tato instance je zachovávána v paměti a využívána v dalších požadavcích. Proto také před čtením souboru XML kontrolujeme, zda `_MyUsers` obsahuje nějaké hodnoty.

Když zjistíme, že `_MyUsers` má hodnotu `null`, použijeme objekt typu `XmlDocument` pro přístup k jednotlivým elementům `<User>` v dokumentu. U každého elementu `<User>` v dokumentu přiřadíme hodnoty instanci `MembershipUser`. Objekt typu `MembershipUser` přebírá následující argumenty:

```
MembershipUser(
  providerName As String, _
  name As String, _
  providerUserKey As Object, _
  email As String, _
  passwordQuestion As String, _
  comment As String, _
  isApproved As Boolean, _
  isLockedOut As Boolean, _
  creationDate As DateTime, _
  lastLoginDate As DateTime, _
  lastActivityDate As DateTime, _
  lastPasswordChangedDate As DateTime, _
  lastLockoutDate As DateTime)
```

I když nezadáváme hodnotu úplně všech položek v této konstrukci, skutečně potřebné hodnoty se přebírají ze souboru XML prostřednictvím objektu typu `XmlNode`. Jakmile je objekt `MembershipUser` naplněn vším potřebným, je zapotřebí přidat jej do objektu `_MyUsers` takto:

```
MyUsers.Add(mu.UserName.ToLower(), mu)
```

Jak již bylo řečeno, metodu `ReadUserFile()` nyní můžeme používat i v jiných metodách než jen `ValidateUser()`. Pamatujte, že jakmile je kolekce `_MyUsers` naplněná, nemusíme ji znovu naplňovat – zůstává k dispozici dalším metodám. Nyní se podíváme na použití toho, co jsme již v aplikaci ASP.NET získali.

Použití poskytovatele XmlMembershipProvider k přihlášení uživatelů

Jestliže jste se v našem příkladu dostali až sem, nepotřebujete již mnoho a můžete začít třídu `XmlMembershipProvider` využívat. V tomto okamžiku máte datový soubor XML představující všechny uživatele vaší aplikace (tento soubor XML najdete ve výpisu 13.7) i deklaraci `XmlFileProvider` v souboru `web.config` aplikace (úpravy tohoto souboru najdete ve výpisu 13.8). Další nezbytnou položkou je samozřejmě třída `XmlMembershipProvider.vb` nebo `.cs` ve složce `App_Code` aplikace. Jestliže však sestavíte poskytovatele jako knihovnu tříd, musíte zajistit správné odkazování na vzniklý soubor DLL v aplikaci ASP.NET (to znamená umístění souboru DLL do složky `Bin`). Jakmile máte splněno vše právě uvedené, je jednoduché začít poskytovatele používat.

Ukážeme si rychlý příklad. Vytvořte stránku `Default.aspx` obsahující jen text *Jste ověřeni!*

Pak vytvořte stránku `Login.aspx` a umístěte na ni jediný serverový ovládací prvek Login. Není nutné provádět na této stránce žádné další změny. Uživatelé se nyní budou moci přihlásit k aplikaci.

> *Další informace o systému členství včetně podrobného vysvětlení různých jím nabízených serverových ovládacích prvků najdete v kapitole 16.*

Až budete mít ve své minimalistické webové aplikaci ASP.NET oba soubory, je zapotřebí provést menší změny v souboru `web.config`, konkrétně povolit formulářové ověřování a zakázat všem anonymním uživatelům zobrazování stránek. Potřebný kód najdete ve výpisu 13.17.

Výpis 13.17: Jak zamezit uživatelům zobrazování aplikace pomocí souboru web.config

```
<configuration>
  <system.web>

    <authentication mode="Forms"/>
    <authorization>
      <deny users="?"/>
    </authorization>

    <!-- Další nastavení vynechána -->

  </system.web>
</configuration>
```

Nyní spusťte stránku `Default.aspx` a uvidíte, že budete hned přesměrováni na `Login.aspx` (tu byste měli ve své aplikaci mít a navíc by měla obsahovat jediný serverový ovládací prvek Login), kam lze zadat některou z kombinací uživatelského jména a hesla podle souboru XML. Ano, je to velmi jednoduché!

Na modelu poskytovatelů v ASP.NET 3.5 je hezké to, že ovládací prvky využívající poskytovatele nezajímá rozdíl mezi podobnými zásadními změnami základního poskytovatele. V našem příkladu jsme odstranili výchozího poskytovatele `SqlMembershipProvider` a nahradili jej poskytovatelem využívajícím soubor XML pouze pro čtení; serverový ovládací prvek Login to v zásadě nezajímá. Jakmile koncový uživatel stiskne přihlašovací tlačítko serverového ovládacího prvku Login, pak se

jen využije metoda `Membership.ValidateUser()`, která pracuje s právě sestaveným poskytovatelem `XmlMembershipProvider`. Jak asi již chápete, jedná se o skutečně silný model.

Rozšíření existujících poskytovatelů

Vlastní poskytovatele nemusíte vytvářet jen z některé z abstraktních bázových tříd, jako je `MembershipProvider`, ale můžete rovněž jednoduše rozšířit některého z existujících poskytovatelů v ASP.NET.

Rádi byste kupříkladu používali systémy členství a správy rolí ve spojení se SQL Serverem, ovšem chcete změnit vnitřní fungování výchozích poskytovatelů (`SqlMembershipProvider` nebo `SqlRoleProvider`). Budete-li pracovat se základním datovým skladem již využívaným některým z připravených poskytovatelů, pak je mnohem rozumnější změnit chování dostupného poskytovatele a nestavět úplně nového od začátku.

Další výhodnou práce s existujícím poskytovatelem je to, že není zapotřebí překrývat vše, co vystavuje. Chcete-li změnit určité chování nějakého vestavěného poskytovatele, pak stačí překrýt jen několik vystavených metod a ostatní ponechat beze změny. Tento přístup je v aplikaci rychle a snadno dosažitelný.

Podívejme se tedy na možnost rozšíření jednoho z vestavěných poskytovatelů v zájmu změny jeho základního fungování.

Omezení schopností rolí novým poskytovatelem LimitedSqlRoleProvider

Předpokládejme, že chcete ve své aplikaci ASP.NET využít nový systém správy rolí a že máte v úmyslu pracovat se SQL Serverem jako se základním datovým skladem. Předpokládejme dále, že chcete rovněž omezit role, které mohou vývojáři ve svých aplikacích vytvářet, a že chcete odstranit schopnost přidávat uživatele do určité role v systému.

Není vhodné začít budovat poskytovatele rolí úplně od začátku implementováním abstraktní třídy `RoleProvider`. Je mnohem lepší odvodit svého poskytovatele od `SqlRoleProvider` a jen změnit chování několika metod zajišťujících vytváření rolí a přidávání uživatelů do rolí.

V tomto příkladu sestavíme poskytovatele přímo v aplikaci jako před chvílí, tedy ve složce `App_Code`. Při skutečném projektu zřejmě využijete knihovnu tříd, protože vytvořený poskytovatel pak může být k dispozici v celé vaší společnosti a vývojové týmy budou moci pracovat s knihovnou DLL a nikoli s měnitelným souborem třídy.

Ve složce `App_Code` vytvořte soubor třídy nazvaný `LimitedSqlRoleProvider.vb` nebo `.cs`. Tato třída musí dědit od `SqlRoleProvider`, čímž získáte strukturu zachycenou ve výpisu 13.18.

Výpis 13.18: Začátek třídy LimitedSqlRoleProvider

VB

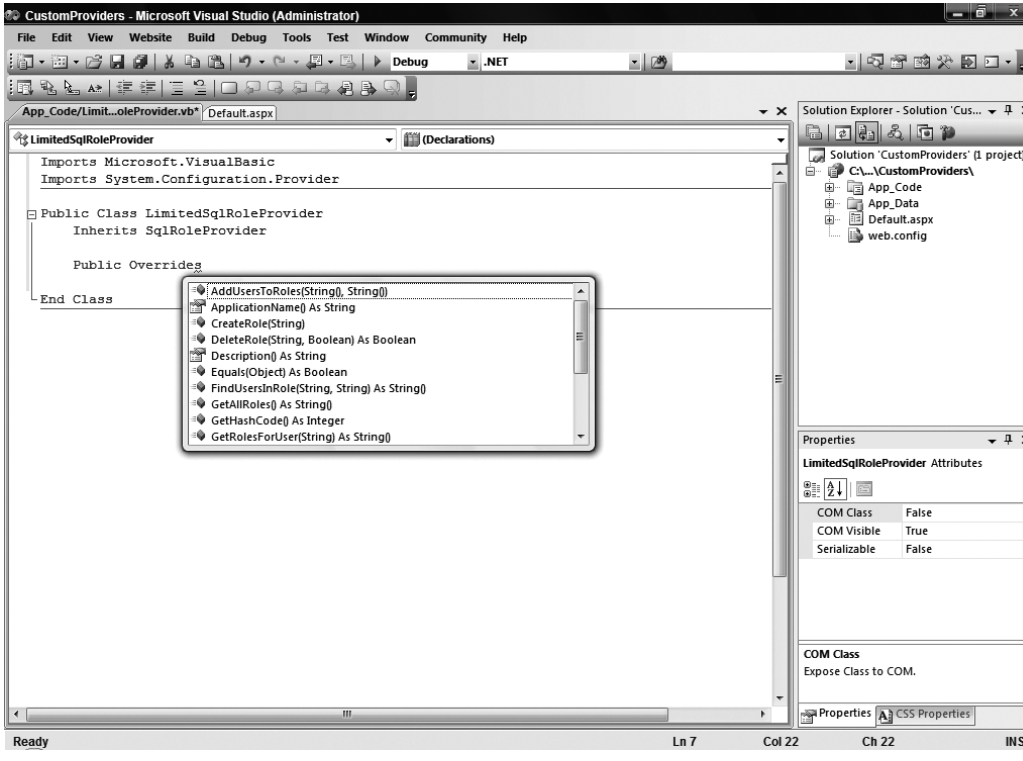
```
Imports Microsoft.VisualBasic
Imports System.Configuration.Provider

Public Class LimitedSqlRoleProvider
```

```
  Inherits SqlRoleProvider
End Class
```

C#

```
using System;
using System.Web;
using System.Web.Security;
using System.Configuration;
using System.Configuration.Provider;

public class LimitedSqlRoleProvider: SqlRoleProvider
{
}
```

To se podobá vytváření třídy `XmlMembershipProvider`. Ovšem v předchozím případě jste mohli využít Visual Studio a nechat si sestavit kompletní kostru třídy se všemi metodami a vlastnostmi, jež bylo zapotřebí k získání funkční nové třídy překrýt. Když se nyní pokusíte ve Visual Studiu o něco podobného, objeví se chyba (při práci s C#) nebo se neobjeví žádný výsledek (při práci ve Visual Basicu), jelikož nepracujete s abstraktní třídou. Není zapotřebí překrývat velké množství metod a vlastností. Protože odvozujete od třídy, jež již dědí od některé z abstraktních tříd, stačí překrýt jen ty metody a vlastnosti, s nimiž potřebujete pracovat, a vše ostatní ponechat beze změny.



Obrázek 13.5

Chcete-li si zobrazit příslušný seznam metod a vlastností ve Visual Studiu, stačí jen zapsat `Public Overrides` (při používání Visual Basicu) nebo `override` (při práci v C#). Funkce IntelliSense následně nabídne dlouhý rozevírací seznam metod a vlastností, s nimiž můžete pracovat (viz obrázek 13.5).

V tomto příkladu překryjeme metody `CreateRole()`, `AddUsersToRoles()` a `DeleteRole()` a to podle následujícího popisu.

Metoda CreateRole()

Metoda `CreateRole()` ve třídě `SqlRoleProvider` umožňuje vývojářům přidat do systému nějakou roli. Jediným požadovaným parametrem této metody je řetězcová hodnota představující název dané role. V tomto příkladu neumožníme vývojářům vytvářet libovolné role, ale poskytovatel je omezí na tvorbu rolí *Administrator* a *Manager*. Toho docílíme v metodě `CreateRole()` kódem odpovídajícím výpisu 13.19.

Výpis 13.19: Jak metodou CreateUser() povolit pouze role Administrator a Manager

VB

```
Public Overrides Sub CreateRole(ByVal roleName As String)
  If (roleName = "Administrator" Or roleName = "Manager") Then
    MyBase.CreateRole(roleName)
  Else
    Throw New _
      ProviderException("Tvorba rolí omezena na Administrator a Manager")
  End If
End Sub
```

C#

```
public override void CreateRole(string roleName)
{
  if (roleName == "Administrator" || roleName == "Manager")
  {
    base.CreateRole(roleName);
  }
  else
  {
    throw new
      ProviderException("Tvorba rolí omezena na Administrator a Manager");
  }
}
```

V metodě je vidět prověření toho, zda je vytvářenou rolí Administrator nebo Manager. Jestliže vytvářená role neodpovídá jedné z povolených, vyvolá se výjimka poskytovatele `ProviderException` informující vývojáře o rolích, které mohou vytvořit.

Je-li rolí Administrator nebo Manager, pak se zavolá metoda `CreateRole()` bázové třídy (`SqlRoleProvider`).

Metoda DeleteRole()

Umožníte-li vývojářům pracujícím s tímto poskytovatelem vytvářet jen konkrétní role, pak jim možná budete chtít také znemožnit odstraňování rolí. V takovém případě budete muset překrýt metodu `DeleteRole()` třídy `SqlRoleProvider` způsobem uvedeným ve výpisu 13.20.

Výpis 13.20: Jak znepřístupnit metodu DeleteRole()

VB

```
Public Overrides Function DeleteRole(ByVal roleName As String, _
  ByVal throwOnPopulatedRole As Boolean) As Boolean
  Return False
End Function
```

C#

```
public override bool DeleteRole(string roleName, bool throwOnPopulatedRole)
{
  return false;
}
```

Z kódu metody `DeleteRole()` je zřejmé, že odstraňování rolí je zcela zakázáno. Místo vyvolání metody `DeleteRole()` bázové třídy kódem

```
Return MyBase.DeleteRole(roleName, throwOnPopulatedRole)
```

se jen vrací hodnota `False` a žádná akce neprobíhá. Další možností je vyvolat výjimku `NotSupportedException` takto:

```
Throw New NotSupportedException()
```

Metoda AddUsersToRoles()

Při procházení metodami, které lze překrýt, si všimněte, že existuje jediná metoda dovolující přidat libovolný počet uživatelů do libovolného počtu rolí. Ve třídě `Roles` tuto metodu využívá několik dalších metod. Ve třídě `Roles` jsou metody jako `AddUserToRole()`, `AddUserToRoles()`, `AddUsersToRole()` a `AddUsersToRoles()`, ovšem všechny nakonec pracují s metodou `AddUsersToRoles()`, jak ji nabízí bázová třída `RoleProvider`.

Předpokládejme, že chcete umožnit vývojářům přidávat uživatele pouze do role Manager, nikoli však do role Administrator. Toho lze dosáhnout konstrukcí metody podobné té ve výpisu 13.21.

Výpis 13.21: Jak zabránit přidávání uživatelů do určité role

VB

```
Public Overrides Sub AddUsersToRoles(ByVal usernames() As String, _
  ByVal roleNames() As String)

  For Each roleItem As String In roleNames
    If roleItem = "Administrator" Then
      Throw New _
        ProviderException("Nemáte oprávnění přidávat uživatele " & _
          " do role Administrator")
    End If
  Next

  MyBase.AddUsersToRoles(usernames, roleNames)
End Sub
```

C#

```
public override void AddUsersToRoles(string[] usernames, string[] roleNames)
{
```

```
  foreach (string roleItem in roleNames)
  {
    if (roleItem == "Administrator")
    {
      throw new ProviderException("Nemáte oprávnění přidávat uživatele" +
        " do role Administrator");
    }
  }

  base.AddUsersToRoles(usernames, roleNames);
}
```

Překrytá metoda iteruje všemi poskytovanými rolemi, a pokud některá z rolí obsažených v poli řetězců odpovídá roli Administrator, vyvolá se výjimka typu `ProviderException` informující vývojáře o tom, že nemůže do této role přidávat žádné uživatele. Přestože to tu není ukázáno, můžete využít stejný přístup také ve spojení s metodou `RemoveUsersFromRoles()` vystavovanou bázovou třídou `RoleProvider`.

Použití nového poskytovatele LimitedSqlRoleProvider

Jakmile máte poskytovatele připraveného k použití, je zapotřebí provést určité úpravy v souboru `web.config`, abyste jej mohli využívat ve své aplikaci ASP.NET. Se změnami v souboru `web.config` se seznamte ve výpisu 13.22.

Výpis 13.22: Provedení potřebných změn v souboru web.config

```
<configuration>
  <system.web>

    <roleManager defaultProvider="LimitedProvider" enabled="true">
      <providers>
        <add connectionStringName="LocalSqlServer" applicationName="/"
          name="LimitedProvider"
          type="LimitedSqlRoleProvider" />
      </providers>
    </roleManager>

  </system.web>
</configuration>
```

Pamatujte, že musíte definovat poskytovatele využívaného aplikací zadáním hodnoty atributu `defaultProvider` a definováním odpovídajícího poskytovatele dále v sekci `<provider>`. Musíte rovněž poskytovatele aktivovat nastavením atributu `enabled` na hodnotu `true`. Systém správy rolí je standardně neaktivní.

S využitím elementu `<add>` je možné doplnit instanci poskytovatele využívající třídu `LimitedSqlRoleProvider`. Protože je tento poskytovatel odvozen od třídy `SqlRoleProvider`, musíte použít některé stejné atributy vyžadované tímto poskytovatelem, jako je atribut `connectionStringName` představující řetězec použitý pro připojení k zadané instanci SQL Serveru.

Když máte připravenou instanci `LimitedSqlRoleProvider` a definovanou v souboru `web.config`, můžete využívat ve své aplikaci třídu `Roles` úplně běžným způsobem. Všimněte si ovšem, že se chování nové třídy výrazně odlišuje od běžného poskytovatele `SqlRoleProvider`.

Vše uvidíte v akci po konstrukci jednoduché stránky ASP.NET zahrnující serverové ovládací prvky TextBox, Button a Label. Tuto stránku najdete ve výpisu 13.23.

Výpis 13.23: Použití Roles.CreateRole()

VB

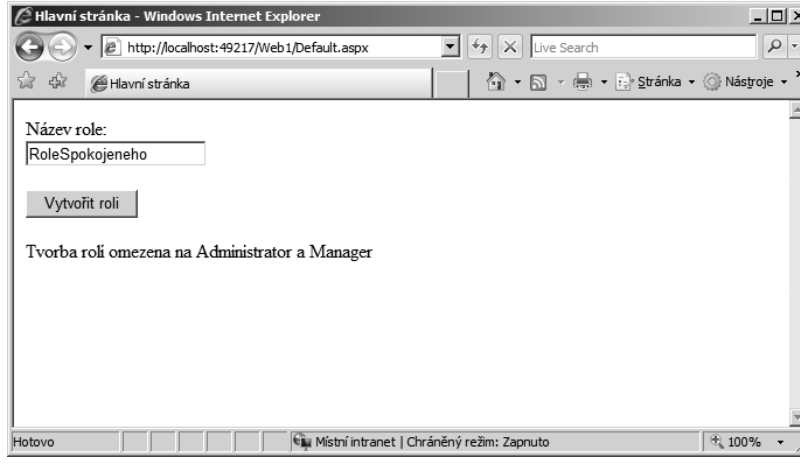
```
<%@ Page Language="VB" %>

<script runat="server">
  Protected Sub Button1_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs)

    Try
      Roles.CreateRole(TextBox1.Text)
      Label1.Text = "Role byla úspěšně vytvořena."
    Catch ex As Exception
      Label1.Text = ex.Message.ToString()
    End Try
  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Hlavní stránka</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    Název role:<br />
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Vytvořit roli"
      OnClick="Button1_Click" /><br />
    <br />
    <asp:Label ID="Label1" runat="server"></asp:Label></div>
  </form>
</body>
</html>
```

C#

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

<script runat="server">
  protected void Button1_Click(object sender, EventArgs e)
  {
    try
    {
      Roles.CreateRole(TextBox1.Text);
```

```
      Label1.Text = "Role byla úspěšně vytvořena.";
    }
    catch (Exception ex)
    {
      Label1.Text = ex.Message.ToString();
    }
  }
</script>
```

Tato jednoduchá stránka .NET vám dovoluje zadat do textového políčka řetězcovou hodnotu a pokusit se vytvořit novou roli s využitím této hodnoty. Všimněte si, že cokoli jiného než role Administrator nebo Manager skončí chybou. Při volání `Roles.CreateRole()` se tato chyba vyvolá, pokud nejsou naplněna pravidla definovaná poskytovatelem. Po zobrazení stránky a zadání jiné role než je Administrator nebo Manager získáte výsledek odpovídající obrázku 13.6.



Obrázek 13.6

Abychom si ukázali poskytovatele v akci, vytvoříme jinou stránku ASP.NET umožňující přidávat uživatele do určitých rolí. Jak již bylo zmíněno, k tomu lze využít různé dostupné metody, náš příklad ovšem pracuje s metodou `Roles.AddUserToRole()`. Kód najdete ve výpisu 13.24.

Výpis 13.24: Pokus o přidání uživatelů do určité role pomocí nového poskytovatele

VB

```
<%@ Page Language="VB" %>

<script runat="server">
  Protected Sub Button1_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs)

    Try
      Roles.AddUserToRole(TextBox1.Text, TextBox2.Text)
      Label1.Text = "Uživatel úspěšně přidán do role"
    Catch ex As Exception
      Label1.Text = ex.Message.ToString()
```

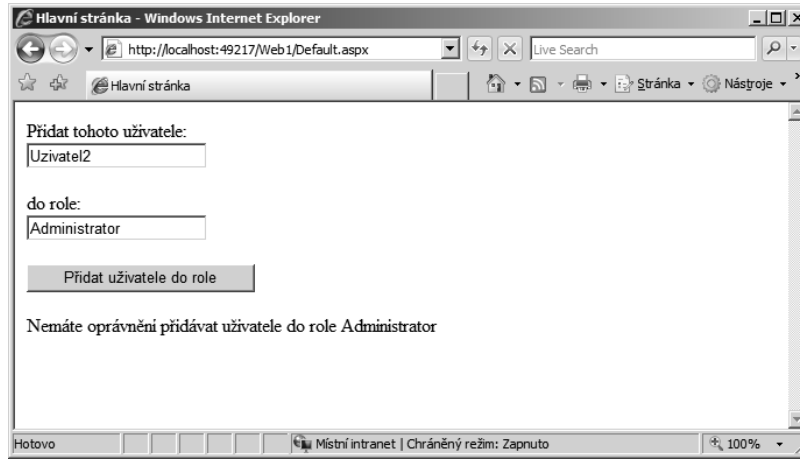
```
    End Try
  End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Hlavní stránka</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    Přidat tohoto uživatele:<br />
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox><br />
    <br />
    do role:<br />
    <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox><br />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Přidat uživatele do role"
      OnClick="Button1_Click" /><br />
    <br />
    <asp:Label ID="Label1" runat="server"></asp:Label></div>
  </form>
</body>
</html>
```

C#

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

<script runat="server">
  protected void Button1_Click(object sender, EventArgs e)
  {
    try
    {
      Roles.AddUserToRole(TextBox1.Text, TextBox2.Text);
      Label1.Text = "Uživatel úspěšně přidán do role";
    }
    catch (Exception ex)
    {
      Label1.Text = ex.Message.ToString();
    }
  }
</script>
```

V tomto příkladu využíváme dvě textové políčka. První se dotazuje na jméno uživatele a druhé na roli, do níž má být přidán. Kód události stisku tlačítka využívá metodu `Roles.AddUserToRole()`. Protože jste poskytovatele sami sestavili, víte, že při pokusu o přidání nějakého uživatele do role Administrator se objeví chyba. Tento pokus ilustruje obrázek 13.7.

V příkladu jsem se pokusil přidat uživatele Uzivatel2 do role Administrator. To samozřejmě vedlo k chybě a k zobrazení zprávy definované v poskytovateli.



Obrázek 13.7

Shrnutí

Z této a předcházející kapitoly byste měli získat dobrý náhled na model poskytovatelů a jeho význam pro dnešní aplikace ASP.NET 3.5. Přestože máte ihned po instalaci k dispozici mnoho poskytovatelů zajišťujících interakci s různými systémy ASP.NET, nejste omezeni jenom na ně. Určitě si můžete sestavit vlastní poskytovatele nebo rozšířit funkčnost poskytovatelů v systému již obsažených.

Tato kapitola zkoumala oba zmíněné scénáře. Nejprve jsme vybudovali vlastního poskytovatele využitelného pro systém členství, v němž jsou data o uživatelích uložena v souboru XML. Následně jsme si prošli příklad rozšíření třídy `SqlRoleProvider` (již existující v ASP.NET) se současnou změnou chování tohoto poskytovatele.