

Kapitel 53

Erweiterung der ASP.NET-Infrastruktur

In diesem Kapitel:

Entwicklung von .NET-Providern	1118
Implementierung eines Mitgliedschaftssystemproviders	1119
Implementierung eines Profileproviders	1122

Die ASP.NET-Infrastruktur lässt sich durch verschiedene Maßnahmen erweitern, insbesondere können folgende Bausteine durch .NET-Klassen implementiert werden und damit bestehende Bausteine ergänzen oder ersetzen:

- HTTP-Module,
- HTTP-Handler,
- Zwischenspeicherabhängigkeiten (CacheDependency),
- Seitenzustandsspeicher,
- Sitzungsprovider,
- Mitgliedschaftsprovider,
- Profilprovider,
- Personalisierungsprovider.

HINWEIS

In diesem Buch kann die Erweiterbarkeit aus Platzgründen nur exemplarisch gezeigt werden.

Entwicklung von .NET-Providern

Mit dem .NET Framework Version 2.0 hat Microsoft ein so genanntes *Providermodell* ergänzt, mit dem ausgewählte Funktionen durch eigene Implementierungen erweitert bzw. vollständig ersetzt werden können. Die meisten in der .NET-Klassenbibliothek enthaltenen Provider dienen ASP.NET. Lediglich der zur Kategorie der FileSettingsProvider gehörende LocalFileSettingsProvider wird in Konsolen- und Windows-Anwendungen verwendet.

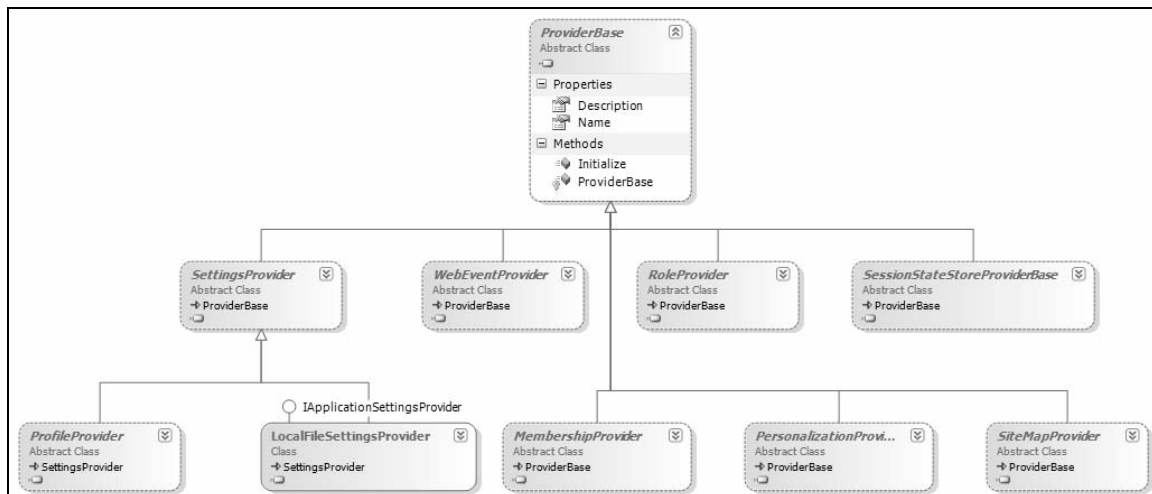


Abbildung 53.1 Provider im .NET-Providermodell

Allgemeine Eigenschaften der Provider

Die Basisklasse `Provider` definiert nur wenige Mitglieder: `Name`, `Description` und `Initialize()`. Der Name eines Providers muss gesetzt sein. Da das Attribut vor Schreibzugriffen geschützt (*read only*) ist, kann ein Setzen jedoch nur im Rahmen einer Überladung von `Initialize()` durch Aufruf der vordefinierten `Initialize()`-Implementierung in der Basisklasse `ProviderBase` erfolgen.

Die Methode `Initialize()` wird von der jeweiligen Umgebung (z.B. dem ASP.NET Page Framework) einmalig aufgerufen und ist dafür zuständig, alle notwendigen Einstellungen des spezifischen Providers zu setzen. Dazu erhält die Methode von der jeweiligen Umgebung eine Menge der in der Anwendungskonfigurationsdatei dem Provider zugeordneten Einstellungen.

```
public override void Initialize(string name, System.Collections.Specialized.NameValueCollection config)
{
    base.Initialize("WWWingsMemberShipProvider", config);
    AppName = config["ApplicationName"];
    TestBetrieb = (config["testBetrieb"] == "true");
    ConnString =
        System.Configuration.ConfigurationManager.ConnectionStrings[config["ConnectionStringName"]].ConnectionString;
}
```

Listing 53.1 Beispiel für die Verwendung von `Initialize()`

Implementierung eines Mitgliedschaftssystemproviders

Ein Mitgliedschaftssystem ist die Schnittstelle zwischen dem ASP.NET-Mitgliedschaftssystem und einem Speicher für Benutzerverwaltungsdaten.

Verfügbare Provider

ASP.NET enthält zwei vordefinierte Mitgliedschaftssystemprovider:

- `ActiveDirectoryMembershipProvider`
- `SqlMembershipProvider`

Eigene Provider benötigt man, wenn man ein anderes Datenbankschema oder Datenbankmanagementsystem bzw. einen anderen Datenspeicher verwenden möchte. Mitgliedschaftssystemprovider sind .NET-Klassen, die von der abstrakten Klasse `System.Web.Security.MembershipProvider` oder vorhandenen Providern (z.B. `SqlMembershipProvider`) erben.

TIPP

Beispiele für die Implementierung eines Mitgliedschaftssystemproviders findet man in dem öffentlich zugänglichen Code zur Implementierung eines `OdbcMembershipProvider` [MSDN02] und `MySqlMembershipProvider` [CPR02].

Konfiguration

Ein Mitgliedschaftssystemprovider wird durch eine Konfigurationssektion in der Datei *web.config* konfiguriert. In der Konfigurationssektion `<membership>` lassen sich neue Provider definieren. Dies erfolgt durch einen Verweis auf den Namen der Klasse, die den Provider implementiert. Neben `name` und `type` kann der Provider beliebige Attribute besitzen; er kann sich, muss sich aber nicht, an die in der Klasse `MembershipProvider` definierten Attribute halten.

```
<!-- Membership -->
<membership
  defaultProvider="WWWingsMemberShipProvider"
  userIsOnlineTimeWindow="15">
  <providers>
    <add
      name="WWWingsMemberShipProvider"
      type="de.WWWings.Web.Provider.WWWingsMemberShipProvider"
      ApplicationName="WWWings"
      testBetrieb="true"
      connectionStringName="CS_WebsiteMDF"
      enablePasswordRetrieval="true"
      enablePasswordReset="true"
      requiresQuestionAndAnswer="true"
      writeExceptionsToEventLog="true" />
    </providers>
  </membership>
```

Listing 53.2 Konfigurationseinstellungen für den World Wide Wings-Membership-Provider

Implementierungsschritte

Die Klasse `MembershipProvider` hat sehr viele Mitglieder, die in einer Unterklasse zu implementieren sind. Tatsächlich muss man aber nicht unbedingt alle Mitglieder implementieren. Zur Unterstützung des Login-Steuerelements reichen folgende Implementierungsdetails:

- Die vom Provider geerbte Methode `Initialize()` ist zu überschreiben: Darin ist die Konfigurationssektion aus der Datei *web.config*, die ASP.NET der Methode `Initialize()` als Parameter übergibt. Außerdem ist der Name des Providers festzulegen. Dies erfolgt durch den Aufruf der Methode in der übergeordneten Klasse: `base.Initialize("WWWingsMemberShipProvider", config)`. Der Provider kann einige der vordefinierten Attribute (z.B. `EnablePasswordRetrieval`, `EnablePasswordReset`, `PasswordFormat`, `MaxInvalidPasswordAttempts`) setzen, muss dies aber nicht.
- Weiterhin sind die für die Anmeldesteuerelemente relevanten Methoden zu überschreiben. Um das Login-Steuerelement zu verwenden, muss nur die Methode `ValidateUser()` überschrieben werden. Andere Methoden wie `GetUser()`, `GetAllUser()`, `CreateUser()`, `DeleteUser()`, `ChangePassword()` und `GetPassword()` sind nur für andere Steuerelemente bzw. das `Membership`-Objekt relevant.

HINWEIS

`Initialize()` wird beim Mitgliedschaftssystemprovider nicht beim Start der Webanwendung, sondern bei der erstmaligen Verwendung (z.B. durch ein Steuerelement) aufgerufen. Während der Laufzeit der Anwendung erfolgt dann aber kein erneuter Aufruf dieser Methode.

TIPP In einigen Punkten hilft die Membership-Klasse bei der Implementierung des Mitgliedschaftsproviders. Insbesondere ist es möglich, ein Kennwort unter Verwendung des in `<machineKey>` hinterlegten symmetrischen Schlüssels zu verschlüsseln (Methoden `EncryptPassword()` und `DecryptPassword()`).

Beispiel

Das folgende Listing zeigt einen Ausschnitt aus der Implementierung eines einfachen Mitgliedschaftssystemproviders, der als Datenbasis eine Tabelle *B_Benutzer* mit den Feldern *B_ID* und *B_Kennwort* verwendet. Das Kennwort wird unverschlüsselt abgelegt. Als besondere Eigenschaft besitzt der Provider einen *TestBetrieb* (gleichnamige Konfigurationseinstellung). Hierüber kann im laufenden Betrieb gesteuert werden, dass alle Anmeldeversuche als erfolgreich gewertet werden – auch wenn das Kennwort nicht stimmt oder es den Benutzer gar nicht gibt.

Die Implementierung beschränkt sich auf die Anbindung an das Login-Steuerelement. Dazu sind lediglich zwei Methoden zu realisieren:

- `Initialize()` zum Übertragen der Konfigurationseinstellungen an das Objekt,
- `ValidateUser()` zur Prüfung von Benutzername und Kennwort.

```
namespace de.WWWings.Web.Provider
{
    /// <summary>
    /// Summary description for WWWingsMemberShipProvider
    /// </summary>
    public class WWWingsMemberShipProvider : System.Web.Security.MembershipProvider
    {
        public WWWingsMemberShipProvider()
        {
            System.Web.HttpContext.Current.Trace.Write("Memhershhip instanziiert!");
        }

        private bool TestBetrieb;
        private string AppName;
        private string ConnString;

        public override void Initialize(string name, System.Collections.Specialized.NameValueCollection config)
        {
            base.Initialize("WWWingsMemberShipProvider", config);
            AppName = config["ApplicationName"];
            TestBetrieb = (config["testBetrieb"] == "true");
            ConnString =
                System.Configuration.ConfigurationManager.ConnectionStrings[config["ConnectionStringName"]].ConnectionString;
        }
        ...

        public override bool ValidateUser(string username, string password)
        {
            // Hack zu Testzwecken
            if (TestBetrieb) return true;

            // Eigentliche Implementierung
            System.Data.SqlClient.SqlConnection conn = new System.Data.SqlClient.SqlConnection();
```

```
conn.ConnectionString = this.ConnString;
conn.Open();
System.Data.SqlClient.SqlCommand cmd = new System.Data.SqlClient.SqlCommand();
cmd.CommandType = CommandType.Text;
cmd.CommandText = "SELECT count(B_ID) FROM B_Benutzer WHERE B_ID = @BID and B_Kennwort = @Kennwort";
cmd.Connection = conn;
System.Data.SqlClient.SqlParameter p1 = new System.Data.SqlClient.SqlParameter("@BID", username);
cmd.Parameters.Add(p1);
System.Data.SqlClient.SqlParameter p2 = new System.Data.SqlClient.SqlParameter("@Kennwort",
    password);
cmd.Parameters.Add(p2);
int Anzahl = (int)cmd.ExecuteScalar();
return (Anzahl == 1);
}
...
}
```

Listing 53.3 Ausschnitte aus der Implementierung eines Membership-Providers

HINWEIS Eine vollständige Implementierung eines Mitgliedschaftssystemproviders kann aufgrund der Länge des Listings nicht im Buch abgedruckt werden. In den Codebeispielen zu diesem Buch finden Sie auch die Implementierung des `OdbcMembershipProvider`, der aus [MSDN02] stammt.

Implementierung eines Profileproviders

Grundsätzlich ist eine andere Form der Speicherung von Profildaten möglich, da die Speicherung auf einem Providermodell basiert. Die in .NET 2.0 mitgelieferte Implementierung für Windows- und Konsolenanwendungen ist der `LocalFileSettingsProvider`. Die in ASP.NET 2.0 verwendete Klasse `SqlProfileProvider` zur Speicherung von Profildaten im Microsoft SQL Server ist ebenso wie der `LocalFileSettingsProvider` von `System.Configuration.SettingsProvider` abgeleitet.

Implementierungsschritte

Die beiden wichtigsten Methoden des Providers sind:

- `GetPropertyValues()`
- `SetPropertyValues()`

`GetPropertyValues()` erhält eine `SettingsPropertyCollection` mit einer Liste von Profildatenelementen, die erwartet werden, und liefert eine `SettingsPropertyValueCollection` an den Aufrufer mit Attribut-Wert-Paaren in Form von `SettingsPropertyValue`-Objekten zurück.

`SetPropertyValues()` erhält `SettingsPropertyValueCollection` mit einzelnen `SettingsPropertyValue`-Objekten, die zu speichern sind.

Beispiel

Ein Beispiel finden Sie unter `/App_Code/Provider/WWWingsProfileProvider`.