Sitecore CMS 6

セキュリティ API クックブック

CMS 開発者のためのコンセプトの概要

目次

Chapter 1 イントロダクション 4
Chapter 2 ユーザー、ドメイン、ロールおよびプロファイル管理 5
2.1 Sitecore セキュリティの概要 6
2.2 Sitecore セキュリティ API の概要 8
2.3 メンバーシップ プロバイダー設定 10
2.4 ログイン フォームのサンプル 12
2.4.1 ASP.NET Login Web コントロールの使用方法 14
2.5 自己登録フォームのサンプル 16
2.5.1 ASP.NET CreateUserWizard Web コントロールの使用 20
2.6 パスワード回復フォームのサンプル 22
2.7 仮想ユーザー 26
2.7.1 仮想ユーザーの作成方法 26
Chapter 3 ユーザー プロファイル 27
3.1 ユーザー プロファイルの概要 28
3.2 標準ユーザー プロファイル プロパティへのアクセス方法 29
3.3 カスタム ユーザー プロファイル プロパティへのアクセス方法 30
3.4 デフォルト ユーザー プロファイルの拡張方法 32
3.5 カスタム ユーザー プロファイルの実装 33
3.5.1 カスタム ユーザー プロファイルの作成方法 33
3.5.2 ユーザー マネージャーを使用したカスタム ユーザー プロファイルの適用方法 34
3.5.3 API を使用したカスタム ユーザー プロファイルの適用方法 34
3.5.4 カスタム ユーザー プロファイル クラスの実装方法 35
3.6 ユーザー プロファイル管理フォームのサンプル 36
3.6.1 ASP.NET ChangePassword Web コントロールの使用方法 39
Chapter 4 アクセス権管理 40
4.1 アクセス権の概要 41
4.2 ユーザー スイッチャー 42
4.3 セキュリティ ディセーブラー 44
4.4 アクセス権の適用 45
Chapter 5 System.Web.Security API 47
5.1 System.Web.Security.Roles 48
5.1.1 System.Web.Security.Roles.CreateRole() 48
5.1.2 System.Web.Security.Roles.DeleteRole() 48
5.2 System.Web.Security.MembershipUser 49
5.2.1 System.Web.Security.MembershipUser.GetUser() 49

5.2.2 System.Web.Security.MembershipUser.ChangePassword() 49
5.2.3 System.Web.Security.MembershipUser.ChangePasswordQuestionAndAnswer() 49
5.2.4 System.Web.Security.MembershipUser.ResetPassword() 50
5.2.5 System.Web.Security.MembershipUser.UnlockUser() 50
5.3 System.Web.Security.Membership 51
5.3.1 System.Web.Security.Membership.GetUserNameByEmail() 51
5.3.2 System.Web.Security.Membership.FindUsersByEmail() 51
Chapter 6 付録 A 52
6.1 Sitecore.Security.AccessControl.AccessRight 53

Chapter 1

イントロダクション

この文書では、ユーザー管理、認証、認可およびユーザー プロファイル管理などの一般的なセキュリティ要件をサポートする Sitecore API のサンプル コードを紹介します。この文書では、使用されるすべての API を説明するのではなく概念の概要を示します。API メソッドの詳細については、API 用のドキュメントでご説明します。[1] 読者の便宜上、この文書では、ASP.NET フレームワークによって提供される一部のセキュリティ API について説明しますが、これらは Sitecore によって一切抽象化されていません。[2]

この文書には次の章があります:

- Chapter 1 – イントロダクション
- Chapter 2 – ユーザー、ドメイン、ロールおよびプロファイル管理
- Chapter 3 – ユーザー プロファイル
- Chapter 4 – アクセス権管理
- Chapter 5 – System.Web.Security API
- Chapter 6 – 付録 A

[1] Sitecore API 文書へのアクセスについては、
http://sdn.sitecore.net/Reference/Sitecore%206/Sitecore_6_API_Reference.aspx を参照してください。
[2] System.Web.Security API の詳細については、
http://msdn.microsoft.com/en-us/library/system.web.security.aspx を参照してください。

Chapter 2

ユーザー、ドメイン、ロールおよびプロファイル管理

この章では、ユーザー、ドメイン、ロールおよびユーザー プロファイルについて説明するとともに、認証 (ログイン)、自己登録およびパスワード管理を含む、ロールとユーザーを管理するためのサンプル コードを示します。ユーザー プロファイルの詳細については、「Chapter 3 ユーザー プロファイル」を参照してください。

この章には次のセクションがあります:

- Sitecore セキュリティの概要
- Sitecore セキュリティ API の概要
- メンバーシップ プロバイダー
- ログイン フォームのサンプル
- 自己登録フォームのサンプル
- パスワード回復フォームのサンプル
- 仮想ユーザー

2.1 Sitecore セキュリティの概要

Sitecore ユーザーとは、システムにアクセスする個人のことを指します。

各ユーザーには、氏名や電子メール アドレスなどのユーザー プロパティを定義するプロファイルがあります。Sitecore では、すべてのユーザーにデフォルト プロファイルを提供します。また、カスタム ユーザー プロファイルを実装することにより、カスタム ユーザー プロパティをより簡単に管理することもできます。

ロールは、ユーザーとネストされたロールのコレクションです。各ユーザーは、任意の数のロールのメンバーになることができます。ユーザーには、ネストされたロールを含む各ロールへのアクセス権があります。

各ロールは、任意の数のロールのメンバーになることができます。Sitecore では、他のロールを含むロールのことをターゲット ロールと呼び、ロールに含まれるロールのことをネストされたロールと呼びます。ユーザーには各ロールへのアクセス権がありますが、それと同様に、ネストされたロールには、そのロールを含む各ターゲット ロールへのアクセス権があります。ユーザーには、ネストされた各ロールを含む各ターゲット ロールへのアクセス権があります。ネストされた各ロールは、ターゲット ロールのメンバーです。

Sitecore ドメインは、ユーザーとロールのコレクションです。ほとんどのユーザーとロール（アカウント）は、ドメイン内に存在します。Sitecore ソリューションには、任意の数のドメインを含めることができます。

アカウント名は、バックスラッシュ文字（"\"）で区切られたドメインとローカル ユーザー名で構成されています。たとえば、ユーザー名 `domain\username` の場合、`domain` 部分はドメイン名を表し、`username` はローカル ユーザー名を表します。ユーザー名に対する参照はすべてドメインを含む必要があります。

Sitecore は、CMS ユーザーおよびロールに関する情報が含まれる Sitecore ドメインと、公開された Web サイトのユーザー ロールに関する情報が含まれる Extranet ドメインの 2 つのドメインが設定された状態で出荷されます。

Sitecore では、論理 Web サイトごとにコンテキスト ドメインが関連付けられます。たとえば、デフォルトの公開された Web サイトに関連付けられたコンテキスト ドメインは Extranet ドメインであり、CMS ユーザー インターフェースに関連付けられたコンテキスト ドメインは Sitecore ドメインです。

Sitecore API にアクセスするすべてのコードは、Sitecore ユーザーのコンテキストで実行されます。認証のために Sitecore API を呼び出さない場合、コードは匿名ユーザーのコンテキストで実行されます。匿名ユーザーは個人ユーザーとして機能しますが、認証されていないユーザーのクラス全体を表します。

コードが認証なしで Sitecore API にアクセスする場合、このコードは Extranet ドメインで匿名ユーザーのコンテキストで実行されます。Sitecore ドメインではユーザーとして認証なしで CMS にアクセスすることはできないため、CMS ユーザー インターフェース内から Sitecore API にアクセスするコードは Sitecore ドメイン内で特定のユーザーとして実行されます。

メモ

Sitecore Extranet ドメインにより、認証が関与する任意の公開された Web サイトを表すことができます。これは、エクストラネットという用語の他の定義より包括的である場合があります。公開された Web サイトで認証を有効にするために追加ドメインを実装する必要はありません。

重要

特に指定のない限り、アカウント名が含まれるすべての文字列にはドメイン名を組み込む必要があります。たとえば、完全なアカウント名 `domain\account` の場合、`domain` はドメイン名を表し、`account` はアカウント名を表します。

重要

ドメインには、アクセス権設定ではなくアカウント定義が含まれます。公開ターゲット データベースにアクセス権の変更が表示されるようにするには、影響を受けるアイテムを公開する必要があります。

2.2 Sitecore セキュリティ API の概要

Sitecore セキュリティ モデルは、元になる .NET メンバーシップ、ロールおよびプロファイル プロバイダーの一部の機能を抽出します。[3] 多くの場合、Sitecore API メソッドまたはメソッドを .NET フレームワークで直接使用しても同じ機能を実現することができます。たとえば、Sitecore API を起動してユーザーを認証するカスタム ソリューションを開発したり、ASP.NET メンバーシップ コントロールとともに Sitecore ソリューションを使用することができます。[4]

Sitecore では、元になる .NET セキュリティ モデルによって提供されるすべての機能について抽象化は行いません。元になる .NET フレームワークによって提供されるメソッドおよびコントロールは、オブジェクトの代わりに文字列を受け入れるもののように、Sitecore API メソッドより若干効率的な場合があります。既にオブジェクトが作成されている場合、Sitecore API メソッドの方が便利な可能性がありますが、パフォーマンスの差異はごくわずかです。

セキュリティに関連する Sitecore API には、次が含まれます:[5]

- **`Sitecore.Context`**: コンテキスト ユーザーを指定するプロパティ、およびそのユーザーが認証されているかどうかが含まれます。

- **`Sitecore.Security.Domains.Domain`**: Sitecore セキュリティ ドメインを表します。セキュリティ ドメインには、ユーザーおよびロール定義が含まれます。これには、ユーザー プロファイルおよびパスワード (デフォルトでは暗号化されています) が含まれます。Sitecore セキュリティ ドメインは、Windows または Active Directory ドメインと同じように機能します。Sitecore は、複数のドメインをサポートしています。デフォルトのセキュリティ ドメインには、CMS ユーザーおよびロールが含まれる Sitecore ドメインと、Web サイト ユーザーおよびロールが含まれる Extranet ドメインが含まれます。各アカウントはドメインのメンバーです。

- **`Sitecore.Security.Accounts.Account`**: `Sitecore.Security.Accounts.Role` および `Sitecore.Security.Accounts.User` のベース クラスです。

- **`Sitecore.Security.Accounts.Role`**: ユーザーおよびメンバー ロールの指定されたコレクションを表します。

- **`Sitecore.Security.Accounts.UserRoles`**: ユーザーに関連付けられたロールのコレクションを表します。

---

[3] .NET プロバイダー モデルの詳細については、http://msdn.microsoft.com/en-us/library/tw292whz.aspx および http://msdn.microsoft.com/en-us/library/aa479030.aspx を参照してください。

[4] ASP.NET メンバーシップ コントロールの詳細については、http://msdn.microsoft.com/en-us/library/ms178329.aspx を参照してください。

[5] Sitecore API 文書へのアクセスについては、http://sdn.sitecore.net/Reference/Sitecore%206/Sitecore_6_API_Reference.aspx を参照してください。

- **Sitecore.Security.Accounts.RolesInRolesManager**: ネストされたロールを処理するためのメソッドを提供します。
- **Sitecore.Security.Accounts.User**: 指定されたユーザーを表します。
- **Sitecore.Security.UserProfile**: 指定されたユーザーのプロファイルを表します。
- **Sitecore.Security.Authentication.AuthenticationManager**: 認証のメソッドを提供します。
- **Sitecore.Security.Authentication.AuthenticationHelper**: 認証の追加メソッドを提供します。
- **Sitecore.Security.Accounts.UserSwitcher**: 別のユーザーのコンテキストでコードを実行します。
- **Sitecore.SecurityModel.SecurityDisabler**: 管理権限を持つユーザーのコンテキストでコードを実行します。
- **Sitecore.Security.AccessControl.AuthorizationManager**: アクセス権を決定および適用するためのメソッドが含まれます。
- **Sitecore.Security.AccessControl.AccessRight**: アクセス権を表します。
- **Sitecore.Security.AccessControl.AccessPermission**: アクセス権の権限状態を表します。
- **Sitecore.Security.AccessControl.PropagationType**: アクセス権をアイテムの子孫に適用するためのルールを表します。
- **Sitecore.Security.AccessControl.AccessRuleCollection**: アイテムのアクセス ルールのコレクションを表します。

- **Sitecore.Security.AccessControl.AccessRuleCollectionHelper**: アクセス権を処理するためのメソッドを提供します。

メモ

Sitecore では、元になる ASP.NET メンバーシップ、ロールおよびプロファイル プロバイダーによって提供されるすべてのセキュリティ機能を抽出するわけではありません。ASP.NET プロバイダー フレームワーク内の関連するメソッドの詳細については、「Chapter 5 System.Web.Security API」を参照してください。[6]

[6] System.Web.Security API の詳細については、
http://msdn.microsoft.com/en-us/library/system.web.security.aspx を参照してください。

2.3 メンバーシップ プロバイダー設定

Sitecore セキュリティは、ASP.NET のメンバーシップ プロバイダー、ロール プロバイダー、およびプロファイル プロバイダーに依拠しています。[7]

メモ
Sitecore では ASP.NET メンバーシップ、ロールおよびプロファイル プロバイダーが使用されるため、Sitecore API を起動するコードを作成する代わりに、ASP.NET Login コントロールを使用することができます。[8] この文書では、必要に応じて特定の ASP.NET Login コントロールを参照しています。

`web.config` 内の適切な `/configuration/system.web/membership/providers/add` エレメントを使用して、パスワードのリセットおよび回復用のオプションを含む、ASP.NET メンバーシップ プロバイダーを設定することができます。[9] `name` `sitecore` がある `<add>` エレメントの `realProviderName` 属性の値と等しい `name` 属性を持つ `<add>` エレメントを更新します。Microsoft SQL サーバー インストールの場合、`realProviderName` は `sql` です。この場合は、`name` 属性 `sql` を持つ `<add>` エレメントを更新します。このエレメントには次の属性を指定することができます:

- **`enablePasswordReset`** (`true` または `false`): パスワードが設定可能かどうか。
- **`enablePasswordRetrieval`** (`true` または `false`): パスワードが検索可能かどうか。`passwordFormat` が `clear` である必要があります。
- **`maxInvalidPasswordAttempts`** (正数): ユーザーがシステムからロックアウトされる前に許可される無効なパスワード試行の回数。
- **`minRequiredNonalphanumericCharacters`** (`0` または正数): 必要な特殊文字の数。
- **`minRequiredPasswordLength`** (正数): 最小限必要なパスワードの長さ。

---

[7] Sitecore による ASP.NET メンバーシップ プロバイダーの使用の詳細については、http://sdn.sitecore.net/Articles/Security/Low_level_Sitecore_Security_and_Custom_Providers.aspx を参照してください。ASP.NET メンバーシップ プロバイダーの詳細については、http://msdn.microsoft.com/en-us/library/aa479031.aspx を参照してください。ASP.NET ロール プロバイダーの詳細については、http://msdn.microsoft.com/en-us/library/aa479032.aspx を参照してください。ASP.NET プロファイル プロバイダーの詳細については、http://msdn.microsoft.com/en-us/library/aa479035.aspx を参照してください。

[8] ASP.NET Login コントロールの詳細については、http://msdn.microsoft.com/en-us/library/ms178329.aspx を参照してください。

[9]ASP.NET メンバーシップ プロバイダー設定属性の詳細については、http://msdn.microsoft.com/en-us/library/whae3t94.aspx を参照してください。

- **passwordFormat** (`Clear`、`Encrypted` または `Hashed`): パスワードのストレージ フォーマット。[10]
- **passwordStrengthRegularExpression** (`string`): パスワードは指定した正規表現と一致する必要があります。
- **requiresQuestionAndAnswer** (`true` または `false`): パスワードをリセットするには、ユーザーのプロファイルにストアされている質問に対する正しい回答が必要です。

- **requiresUniqueEmail** (`true` または `false`): 各ユーザーに一意の電子メール アドレスが必要であるかどうか。

メモ

`passwordFormat` が `Encrypted` である場合、.NET では、`machine.config` 内の暗号化キーが使用されます。これは、これらの暗号化パスワードを使用して認証する必要があるすべてのシステムで同じ値を使用して設定する必要があります。

デフォルトの Sitecore 設定では、ユーザーがパスワードを指定できない場合、パスワードの回復はサポートされません。ユーザーのパスワードをランダム文字列で構成される一時的パスワードにリセットし、ユーザーがこのパスワードを使用できるようにするには、`System.Web.Security.MembershipUser.ResetPassword()` メソッドを使用します。これにより、ユーザーは喪失したパスワードを回復しなくても、このランダム パスワードを使用してログインし、そのパスワードを変更することができます。

[10] ASP.NET パスワード ストレージ フォーマットの詳細については、http://msdn.microsoft.com/en-us/library/system.web.security.membershipprovider.passwordformat.aspx を参照してください。

2.4 ログイン フォームのサンプル

デフォルトでは、Extranet ドメインを使用することにより、ユーザーを認証するためのコードビハインドがあるログイン フォームを収容するサブレイアウトを実装することができます。

メモ
API を直接起動する代わりに、ASP.NET Login Web コントロールを使用して公開された Web サイトを認証することができます。[11] ASP.NET Login Web コントロールの詳細については、次のセクションの「ASP.NET Login Web コントロールの使用方法」を参照してください。

次のサブレイアウト ファイルのサンプル コードでは、非常に簡単なログイン データ エントリ フォームを実装しています:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="Login.ascx.cs"
  Inherits="Namespace.Web.UI.Login" %>

Username: <asp:textbox id="txtUsername" runat="server" /><br />
Password: <asp:textbox id="txtPassword" runat="server" textmode="password" /><br />
Persistent: <asp:checkbox id="chkPersist" runat="server" /><br />
<asp:button id="btnGo" runat="server" Text="Go" /><br />
<asp:label id="lblMessage" runat="server" />
```

このログイン フォームは、次のもので構成されています:

- ユーザーがユーザー名を入力するためのテキスト フィールド。
- ユーザーがパスワードを入力するためのパスワード フィールド。
- 認証が永続的であるか、あるいはこのセッションに対してのみ適用されるかを制御するためのチェックボックス。
- フォームをサブミットするためのボタン。
- フォームのサブミットの結果として生じるエラー メッセージを収容するためのラベル。

メモ
認証は、デフォルトではセッションに対して適用され、永続的なものではありません。セッション認証の場合、セッション Cookie が作成されます。セッション認証の場合、ユーザーがブラウザー ウィンドウを閉じた後にこの Web サイトに戻るには、認証用のユーザー名とパスワードを再入力する必要があります。永続的認証の場合、ブラウザー Cookie が作成されます。永続的認証の場合、ユーザーが Web サイトに戻るときにユーザー名とパスワードを再入力する必要がありません。

[11] ASP.NET Login Web コントロールの詳細については、
http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx を参照してください。

次のサブレイアウト コードビハインド ファイルのサンプル コードでは、ユーザーを認証するためのロジックを実装しています:

```
using System;

namespace Namespace.Web.UI
{
  public partial class Login : System.Web.UI.UserControl
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      if (IsPostBack)
      {
        if (String.IsNullOrEmpty(txtUsername.Text))
        {
          lblMessage.Text = "Invalid username.";
        }
        else if (String.IsNullOrEmpty(txtPassword.Text))
        {
          lblMessage.Text = "Invalid password.";
        }
        else
        {
          try
          {
            Sitecore.Security.Domains.Domain domain = Sitecore.Context.Domain;
            string domainUser = domain.Name + @"\" + txtUsername.Text;

            if (Sitecore.Security.Authentication.AuthenticationManager.Login(domainUser,
              txtPassword.Text, chkPersist.Checked))
            {
              Sitecore.Web.WebUtil.Redirect("/");
            }
            else
            {
              throw new System.Security.Authentication.AuthenticationException(
                "Invalid username or password.");
            }
          }
          catch (System.Security.Authentication.AuthenticationException)
          {
            lblMessage.Text = "Processing error.";
          }
        }
      }
    }
  }
}
```

このコードの背後にあるロジックは、次のとおりです:

1. ページがポストバックしていない場合、ユーザーにはデータをフォームに入力する機会がありません。この場合、何も行いません。
2. ユーザーによってユーザー名が入力されていない場合、エラー メッセージを表示し、それ以上の処理は行いません。
3. ユーザーによってパスワードが入力されていない場合、エラー メッセージを表示し、それ以上の処理は行いません。
4. コンテキスト ドメイン内の完全なユーザー名と、ユーザーが入力したユーザー名を確認します。

5. 完全なユーザー名と、ユーザーが入力したユーザー名およびパスワードを使用してユーザーを認証できる場合、ホーム ページにリダイレクトし、それ以上の処理は行いません。

6. ユーザーを認証できない場合、例外をスローします。これにより、汎用エラー メッセージを表示し、それ以上の処理は行いません。

重要
前のコードの例外管理ロジックに注意してください。このコードでは、例外の原因とは関係なく汎用エラー メッセージを出力しています。ユーザー名とパスワードが無効であることを明確に示すメッセージを出力すると、アタッカーがユーザー名を収集できてしまう可能性があるため、このようなメッセージは出力しないようにしてください。ログイン フォームに対する SSL 暗号化や、少なくとも HTTP を介して送信されるパスワードの暗号化を検討してください。認証用の第 3 係数の使用を検討してください。

メモ
ユーザーをログアウトするには、`Sitecore.Security.Authentication.AuthenticationManager.Logout()` を呼び出します。

2.4.1 ASP.NET Login Web コントロールの使用方法

ASP.NET Login Web コントロールを使用してユーザーを認証することができます。[12] このコントロールを使用する場合、ユーザーが含まれるドメインを検討する必要があります。たとえば、ASP.NET Login Web コントロールをサブレイアウトに追加します:

```
<asp:login id="loginControl" runat="server" DestinationPageUrl="/" />
```

次に基づくサブレイアウトにコードビハインドを追加します。これにより、ユーザー名にコンテキスト ドメインを追加するが、ユーザーによって入力されたパスワードが無効である場合はコンテキスト ドメインを削除し、ユーザーにドメイン名が表示されないようにするロジックを提供します:

```
using System;

namespace Namespace.Web.UI
{
  public partial class LoginForm : System.Web.UI.UserControl
  {
    private string  usernameAsEntered = String.Empty;

    protected override void OnInit(EventArgs e)
    {
      base.OnInit(e);
      loginControl.LoggingIn += new LoginCancelEventHandler(this.Login LoggingIn);
      loginControl.LoginError += new EventHandler(this.Login LoginError);
    }

    private void Login_LoggingIn(object sender, LoginCancelEventArgs e)
    {
```

[12] ASP.NET Login Web コントロールの詳細については、
http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx を参照してください。

```
            string domainUser = Sitecore.Context.Domain.GetFullName(loginControl.UserName);

            if (System.Web.Security.Membership.GetUser(domainUser) != null)
            {
               usernameAsEntered = loginControl.UserName;
              loginControl.UserName = domainUser;
            }
          }

          private void Login LoginError(object sender, EventArgs e)
          {
            loginControl.UserName = _usernameAsEntered;
          }
        }
      }
```

このコードの背後にあるロジックは、次のとおりです:

1. ユーザーによって ASP.NET Login コントロールがサブミットされたら、ユーザーによって入力されたユーザー名にコンテキスト ドメイン名とバックスラッシュ文字を追加することによって完全なユーザー名を確認します。このユーザーが存在する場合、ユーザーによって入力された元のユーザー名をストアし、ユーザー名入力フィールドの値を、ドメインを含む完全なユーザー名に設定します。これにより、ユーザーを認証する ASP.NET Login Web コントロールにドメイン名を提供します。

2. ユーザーを認証できない場合、ユーザー名入力フィールドの値を、ユーザーによって入力された値にリセットします。これにより、コンテキスト ドメインを削除し、ユーザーによってドメインが認識されないようにします。

2.5 自己登録フォームのサンプル

多くの Web サイトでは、ユーザーによる登録を許可しています。これにより、ユーザー名、パスワードおよびロール メンバーシップなどのユーザー プロファイル プロパティをユーザーに関連付けます。通常、登録により、追加コンテンツや他の機能に対するアクセス権をユーザーに付与します。新しい Web サイト ユーザーを登録するためのコードビハインドがある自己登録フォームを収容するサブレイアウトを実装することができます。

メモ

API を直接起動する代わりに、ASP.NET CreateUserWizard Web コントロールを使用してユーザーを認証することができます。[13] ASP.NET CreateUserWizard Web コントロールの使用の詳細については、次のセクションの「ASP.NET CreateUserWizard Web コントロールの使用」を参照してください。

メモ

`Sitecore.Security.Accounts.User.Delete()` メソッドを呼び出してユーザーを削除します。たとえば、ドメイン `domain` 内のユーザー `user` を削除する方法は、次のとおりです:

```
string domainUser = @"domain\user";

if (Sitecore.Security.Accounts.User.Exists(domainUser))
{
  Sitecore.Security.Accounts.User user =
    Sitecore.Security.Accounts.User.FromName(domainUser,false);
  user.Delete();
}
```

次のサブレイアウト ファイルのサンプル コードでは、非常に簡単な自己登録データ エントリ フォームを実装しています:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="Register.ascx.cs"
  Inherits="Namespace.Web.UI.Register" %>

Username: <asp:textbox id="txtUsername" runat="server" /><br />
Email: <asp:textbox id="txtEmail" runat="server" /><br />
Password: <asp:textbox id="txtPassword" textmode="password" runat="server" /><br />
Confirm: <asp:textbox id="txtPasswordConfirm" textmode="password" runat="server" /><br />
Question: <asp:textbox id="txtQuestion" textmode="password" runat="server" /><br />
Answer:<asp:textbox id="txtAnswer" TextMode="password" runat="server" /><br />
Persistent: <asp:checkbox id="chkPersist" runat="server" /><br />
<asp:Button id="btnGo" text="Go" runat="server" /><br />
<asp:Label id="lblMessage" runat="server" />
```

[13] ASP.NET CreateUserWizard Web コントロールの詳細については、http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.createuserwizard.aspx を参照してください。

この自己登録フォームには、次が含まれます:

- ユーザーが目的のユーザー名を入力するためのテキスト フィールド。
- ユーザーが電子メール アドレス入力するためのテキスト フィールド。
- ユーザーがパスワードを入力するためのパスワード フィールド。
- パスワードを確認するためのパスワード フィールド。これにより、ユーザーがパスワードを誤って入力していないかどうかを確認することができます。
- ユーザーがセキュリティ プロファイルに関する質問を入力するためのテキスト フィールド。
- ユーザーがこのセキュリティ上の質問に対する回答を入力するためのテキスト フィールド。
- ユーザーが認証を永続的にするかこのセッションに対して適用するかを制御するためのチェックボックス。
- フォームをサブミットするためのボタン。
- フォームのサブミットの結果として生じるエラー メッセージを収容するためのラベル。

メモ
ほとんどの実装では、ユーザーはセキュリティに関する質問のテキストを入力しません。通常、ユーザーは事前定義された質問に回答するか、用意されている 1 つまたは複数のリストから 1 つまたは複数の質問を選択します。可能な場合は、質問のテキストではなく質問の ID をストアしてください。質問と回答が複数ある場合は、XML またはパイプ区切りのリストをストアしてください。回答は同じフォーマットでストアしてください。

次のサブレイアウト コードビハインド ファイルのサンプル コードでは、ユーザーを作成するためのロジックを実装しています:

```
using System;

namespace Namespace.Web.UI
{
  public partial class Register : System.Web.UI.UserControl
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      if (IsPostBack)
      {
        this.lblMessage.Text = String.Empty;

        if(String.IsNullOrEmpty(txtUsername.Text))
        {
          lblMessage.Text = "Invalid username.";
        }
        else if(System.Web.Security.Membership.Provider.RequiresUniqueEmail
          && String.IsNullOrEmpty((txtEmail.Text)))
        {
```

```
      lblMessage.Text = "Invalid email address.";
    }
    else if(System.Web.Security.Membership.RequiresQuestionAndAnswer
      && (String.IsNullOrEmpty(txtQuestion.Text)
      || String.IsNullOrEmpty(txtAnswer.Text)))
    {
      lblMessage.Text = "Specify question and answer.";
    }
    else if(String.IsNullOrEmpty(txtPassword.Text)
      || String.IsNullOrEmpty(txtPasswordConfirm.Text))
    {
      lblMessage.Text = "Invalid password.";
    }
    else if(txtPassword.Text!=txtPasswordConfirm.Text)
    {
      lblMessage.Text = "Passwords don't match.";
    }
    else
    {
      string domainUser = Sitecore.Context.Domain.GetFullName(txtUsername.Text);

      try
      {
        if(Sitecore.Security.Accounts.User.Exists(domainUser))
        {
          throw new System.Web.Security.MembershipCreateUserException(
            domainUser + " exists.");
        }
        else if(System.Web.Security.Membership.Provider.RequiresUniqueEmail
          && !String.IsNullOrEmpty(
          System.Web.Security.Membership.Provider.GetUserNameByEmail(txtEmail.Text)))
        {
          throw new System.Web.Security.MembershipCreateUserException(
            txtEmail.Text + " already registered.");
        }
        else
        {
          System.Web.Security.MembershipCreateStatus status;
          System.Web.Security.Membership.CreateUser(domainUser,
            txtPassword.Text, txtEmail.Text, txtQuestion.Text,
            txtAnswer.Text, true, out status);

          if(!status.Equals(System.Web.Security.MembershipCreateStatus.Success))
          {
            throw new System.Web.Security.MembershipCreateUserException(
              status.ToString());
          }

          if(Sitecore.Security.Authentication.AuthenticationManager.Login(
            domainUser, txtPassword.Text, chkPersist.Checked))
          {
            Sitecore.Web.WebUtil.Redirect("/profile.aspx");
          }
          else
          {
            throw new System.Web.Security.MembershipCreateUserException(
              "Unable to login after creating " + domainUser );
          }
        }
      }
      catch(System.Web.Security.MembershipCreateUserException)
      {
        lblMessage.Text = "Processing error.";
      }
    }
```

```
                }
            }
        }
    }
```

このコードの背後にあるロジックは、次のとおりです:

1. ページがポストバックしていない場合、ユーザーにはデータをフォームに入力する機会がありません。この場合、何も行いません。

2. ユーザーによってユーザー名が入力されていない場合、エラー メッセージを表示し、それ以上の処理は行いません。

3. システム設定上、ユーザーごとに一意の電子メール アドレスが必要である際にユーザーによって電子メール アドレスが入力されていない場合、エラー メッセージを表示し、それ以上の処理は行いません。

4. システム設定上、各ユーザーがセキュリティに関する質問と回答を指定する必要がある際にユーザーによってこれらのフィールドの 1 つまたは両方が入力されていない場合、エラー メッセージを表示し、それ以上の処理は行いません。

5. ユーザーによってパスワードが指定されていないかそのプロセスが確認されていない場合、エラー メッセージを表示し、それ以上の処理は行いません。ユーザーによって入力された 2 つのパスワードが等しくない場合、エラー メッセージを表示し、それ以上の処理は行いません。

6. コンテキスト ドメイン内のユーザー名と、ユーザーによって入力されたユーザー名を確認します。このユーザー名が既に存在する場合、例外をスローします。これにより、汎用エラー メッセージを表示し、それ以上の処理は行いません。

7. システム設定上、ユーザーごとに一意の電子メール アドレスが必要である際にユーザーによって一意の電子メール アドレスが入力されていない場合、例外をスローします。これにより、汎用エラー メッセージを表示し、それ以上の処理は行いません。

8. ユーザーの作成を試行します。ユーザーを作成できなかった場合、例外をスローします。これにより、汎用エラー メッセージを表示し、それ以上の処理は行いません。

9. ユーザーをログインします。ユーザーが認証された場合、ユーザー /profile.aspx をリダイレクトしてそのプロファイルを管理します。ユーザーが認証されなかった場合、例外をスローし、汎用エラー メッセージを表示します。

2.5.1 ASP.NET CreateUserWizard Web コントロールの使用

コードビハインドを作成する代わりに ASP.NET CreateUserWizard Web コントロールを使用してユーザーを作成することができます。[14] このコントロールを使用する場合、ユーザーが含まれるドメインを検討する必要があります。

たとえば、ASP.NET CreateUserWizard Web コントロールをサブレイアウトに追加します:

```
<asp:CreateUserWizard runat="server" id="createUserWizardControl"
  ContinueDestinationPageUrl="/" />
```

次に基づくサブレイアウトにコードビハインドを追加します。これにより、ユーザー名にコンテキスト ドメインを追加してユーザーがコンテキスト ドメインを指定しなくてすむようにするが、ユーザーを作成できない場合はコンテキスト ドメインを削除するロジックを提供します:

```
using System;

namespace Namespace.Web.UI
{
  public partial class CreateUser : System.Web.UI.UserControl
  {
    private string  usernameAsEntered = String.Empty;

    private void CreateUserWizard CreatingUser(object sender, EventArgs e)
    {
      string domainUser =
        Sitecore.Context.Domain.GetFullName(createUserWizardControl.UserName);

      if (System.Web.Security.Membership.GetUser(domainUser) == null)
      {
        _usernameAsEntered = createUserWizardControl.UserName;
        createUserWizardControl.UserName = domainUser;
      }
    }

    private void CreateUserWizard CreateUserError(object sender, EventArgs e)
    {
      createUserWizardControl.UserName = _usernameAsEntered;
    }

    protected override void OnInit(EventArgs e)
    {
      base.OnInit(e);
      createUserWizardControl.CreatingUser +=
        new LoginCancelEventHandler(this.CreateUserWizard CreatingUser);
      createUserWizardControl.CreateUserError +=
        new CreateUserErrorEventHandler(this.CreateUserWizard CreateUserError);
    }
  }
}
```

[14] ASP.NET CreateUserWizard Web コントロールの詳細については、http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.createuserwizard.aspx を参照してください。

このコードの背後にあるロジックは、次のとおりです:

1. ユーザーによって ASP.NET CreateUserWizard コントロールがサブミットされたら、ユーザーによって入力されたユーザー名にコンテキスト ドメイン名とバックスラッシュ文字を追加することによって完全なユーザー名を確認します。このユーザーが存在する場合、ユーザーによって入力された元のユーザー名をストアし、ユーザー名入力フィールドの値を、ドメインを含む完全なユーザー名に設定します。これにより、ユーザーを作成する ASP.NET CreateUserWizard Web コントロールにドメイン名を提供します。

2. システムがユーザーを作成できない場合、ユーザー名入力フィールドの値を、ユーザーによって入力された値にリセットします。これにより、コンテキスト ドメインを削除し、ユーザーによってドメインが認識されないようにします。

2.6 パスワード回復フォームのサンプル

ユーザーによるパスワードの回復またはリセットを許可するためのコードビハインドがあるフォームを収容するサブレイアウトを実装することができます。

メモ
API を直接起動する代わりに、ASP.NET PasswordRecovery Web コントロールを使用してユーザーを認証することができます。[15]

重要
パスワード リセットまたは回復フォームは、未認証のユーザーがアクセスできる必要があります。

次のサブレイアウト ファイルのサンプル コードでは、非常に簡単なパスワード回復データ エントリ フォームを実装しています:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="LostPassword.ascx.cs"
  Inherits="Namespace.Web.UI.LostPassword" %>

Username: <asp:textbox id="txtUsername" runat="server" /><br />
Answer: <asp:textbox id="txtAnswer" runat="server" textmode="password" /><br />
<asp:button id="btnGo" runat="server" text="Go" /><br />
<asp:label id="lblMessage" runat="server" /><br />
```

このログイン フォームは、次のもので構成されています:

- ユーザーがユーザー名を入力するためのテキスト フィールド。
- ユーザーがアカウントに関するセキュリティ上の質問に対する回答を入力するためのパスワード フィールド。
- 認証が永続的であるかこのセッションに対してのみ適用されるかを制御するためのチェックボックス。
- フォームをサブミットするためのボタン。
- フォームのサブミットの結果として生じるエラー メッセージを収容するためのラベル。

[15] ASP.NET PasswordRecovery Web コントロールの詳細については、http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.passwordrecovery.aspx および http://msdn.microsoft.com/en-us/library/ms178335.aspx を参照してください。

次のサブレイアウト コードビハインド ファイルのサンプル コードでは、ユーザーのパスワードを回復またはリセットするためのロジックを実装しています:

```
using System;

namespace Namespace.Web.UI
{
  public partial class LostPassword : System.Web.UI.UserControl
  {
    protected void Page Load(object sender, EventArgs e)
    {
      if(IsPostBack)
      {
        lblMessage.Text = String.Empty;

        if(String.IsNullOrEmpty(txtUsername.Text))
        {
          lblMessage.Text = "Invalid user.";
        }
        else if(System.Web.Security.Membership.RequiresQuestionAndAnswer
          && String.IsNullOrEmpty(txtAnswer.Text))
        {
          lblMessage.Text = "Invalid answer.";
        }
        else
        {
          try
          {
            string domainUser = Sitecore.Context.Domain.GetFullName(txtUsername.Text);

            if(!Sitecore.Security.Accounts.User.Exists(domainUser))
            {
              throw new System.Security.Authentication.AuthenticationException(
                domainUser + " does not exist.");
            }
            else
            {
              System.Web.Security.MembershipUser user =
                System.Web.Security.Membership.GetUser(domainUser);

              if(System.Web.Security.Membership.EnablePasswordRetrieval)
              {
                lblMessage.Text = "Password for " + user.UserName + ": ";

                if(System.Web.Security.Membership.RequiresQuestionAndAnswer)
                {
                  lblMessage.Text += user.GetPassword(txtAnswer.Text);
                }
                else
                {
                  lblMessage.Text += user.GetPassword();
                }
              }
              else if(System.Web.Security.Membership.EnablePasswordReset)
              {
                lblMessage.Text = "New password for " + user.UserName + ": ";

                if(System.Web.Security.Membership.RequiresQuestionAndAnswer)
                {
                  lblMessage.Text += user.ResetPassword(txtAnswer.Text);
                }
                else
```

```
                        {
                          lblMessage.Text += user.ResetPassword();
                        }
                      }
                      else
                      {
                        throw new System.Configuration.ConfigurationErrorsException(
                          "Cannot retrieve or reset passwords.");
                      }
                    }
                  }
                  catch(System.Security.Authentication.AuthenticationException)
                  {
                    lblMessage.Text = "Processing error.";
                  }
                  catch(System.Configuration.ConfigurationErrorsException)
                  {
                    lblMessage.Text = "Configuration error.";
                  }
                }
              }
            }
          }
        }
```

このコードの背後にあるロジックは、次のとおりです:

1. ページがポストバックしていない場合、ユーザーにはデータをフォームに入力する機会がありません。この場合、何も行いません。
2. 無効なデータがあるフォームのサブミットに関するエラー メッセージが含まれる可能性があるラベルをクリアします。
3. ユーザーによってユーザー名が入力されていない場合、エラー メッセージを設定し、それ以上の処理は行いません。
4. システム設定上、ユーザーに関するセキュリティ上の質問に回答する必要がある際にユーザーが回答を入力していない場合、エラー メッセージを設定し、それ以上の処理は行いません。
5. コンテキスト ドメイン内のユーザー名と、ユーザーによって入力されたユーザー名を確認します。
6. ユーザーによって指定されたアカウントが存在しない場合、例外をスローします。これにより、汎用エラー メッセージを表示し、それ以上の処理は行いません。
7. システム設定上、パスワードの回復が可能な場合、ユーザーのパスワードを表示し、それ以上の処理は行いません。
8. システム設定上、パスワードのリセットが可能な場合、ユーザーのパスワードをリセットし、新しいパスワードを表示し、それ以上の処理は行いません。
9. 無効な設定であることを示す例外をスローします。

メモ

ユーザーが無効なパスワードを使用して、`web.config` に定義されている適切なメンバーシップ プロバイダーの `maxInvalidPasswordAttempts` 属性によって許可されている回数を超えて認証しようとすると、ユーザーはシステムからロックアウトされます。このユーザーは、パスワードが喪失した状態でロックアウトを体験する可能性があります。ユーザーのロック解除の詳細については、「System.Web.Security.MembershipUser.UnlockUser()」のセクションを参照してください。

2.7 仮想ユーザー

仮想ユーザーを使用すると、カスタム ASP.NET メンバーシップ プロバイダーを実装しなくてもサードパーティ認証システムを統合することができます。仮想ユーザーでは、サードパーティ システムを介して認証を提供しますが、多くの場合、認証用として Sitecore のロール プロバイダーを使用します。この方法により、セントラル リポジトリ内のユーザーを管理することができますが、Sitecore セキュリティを使用して CMS オーディエンス セグメントおよび他の認可ロールを管理することもできます。

重要
仮想ユーザーは一時的なものです。仮想ユーザーは、ユーザーがログアウトした後にシステム上に長く残ることはありません。仮想ユーザーは、このセクションで説明されているように、他のすべての点においては他のユーザーと似ています。

たとえば、「ログイン フォームのサンプル」のセクションにあるログイン フォームのサンプルと同じような仮想ユーザー用のログイン フォームを実装することができます。Sitecore に基づいてユーザーを認証するのではなく、既存のセキュリティ システムに基づいてユーザーを認証します。次に、後で説明するように、Sitecore API を呼び出して仮想ユーザーを認証します。仮想ユーザーを Sitecore ロールに関連付け、カスタム メンバーシップやロール プロバイダーを実装せずにアクセス権を使用して、仮想ユーザーによる CMS データベース内のデータに対するアクセスを制御することができます。

2.7.1 仮想ユーザーの作成方法

`Sitecore.Security.Authentication.AuthenticationManager.BuildVirtualUser()` メソッドは、最初のパラメーターによって指定されたユーザー名を持つ仮想ユーザーを構築して返します。2 番目のパラメーターは、仮想ユーザーが認証されるかどうかを制御します。たとえば、氏名 `full name` を持つドメイン `domain` 内の仮想ユーザー `user` をドメイン `domain` 内のロール `role` のメンバーとして構築し、このユーザーにログインする方法は、次のとおりです:

```
        Sitecore.Security.Accounts.User user =
          Sitecore.Security.Authentication.AuthenticationManager.BuildVirtualUser(
          @"domain\user",true);

        if(user!=null)
        {
          string domainRole = @"domain\role";

          if(Sitecore.Security.Accounts.Role.Exists(domainRole))
          {
            user.Roles.Add(Sitecore.Security.Accounts.Role.FromName(domainRole));
          }

          Sitecore.Security.UserProfile profile = user.Profile;
          profile.FullName = "full name";
          profile.Save();
          Sitecore.Security.Authentication.AuthenticationManager.Login(user.Name);
        }
```

重要
仮想ユーザーに `Sitecore.Security.UserProfile.Comment` プロパティを設定することはできません。

Chapter 3

ユーザー プロファイル

この章では、ソリューションについて説明し、ユーザー プロファイルをカスタマイズするためのサンプル コードを示します。

この章には次のセクションがあります:

- ユーザー プロファイルの概要
- 標準ユーザー プロファイル プロパティへのアクセス方法
- カスタム ユーザー プロファイル プロパティへのアクセス方法
- デフォルト ユーザー プロファイルの拡張方法
- カスタム ユーザー プロファイルの実装
- ユーザー プロファイル管理フォームのサンプル

3.1 ユーザー プロファイルの概要

Sitecore では、各ユーザーをユーザー プロファイルに関連付けます。デフォルト ユーザー プロファイルには、ユーザーの氏名や電子メール アドレスなどのプロパティが含まれます。Sitecore には、これら特定のユーザー プロファイル プロパティにアクセスするための API や、カスタム プロファイル プロパティにアクセスするための API が用意されています。

デフォルト ユーザー プロファイルを拡張したり、ユーザー マネージャーでカスタム ユーザー プロファイル プロパティを操作するためのフィールドが含まれるカスタム ユーザー プロファイルを実装することができます。デフォルト ユーザー プロファイルを拡張するか、ユーザー マネージャーを介してカスタム ユーザー プロファイル プロパティを操作する機能を提供するカスタム ユーザー プロファイルを実装します。

カスタム ユーザー プロファイル プロパティに対するプログラムからのアクセスを標準化する .NET クラスを実装することができます。

カスタム クラスを使用してユーザー プロファイルを表すためにカスタム ユーザー プロファイルを実装することや、カスタム プロファイル プロパティにアクセスするために ASP.NET プロファイル プロバイダーを実装することを目的として、デフォルト プロファイルを拡張する必要はありません。この場合、次のセクションの「カスタム ユーザー プロファイル プロパティへのアクセス方法」で説明されているメソッドを使用すればすみます。

3.2 標準ユーザー プロファイル プロパティへのアクセス方法

Sitecore では、デフォルトで `Sitecore.Security.UserProfile` を使用してユーザー プロファイルを表します。このクラスには、次のプロパティが含まれます:

- **`FullName`**: ユーザーの氏名
- **`Email`**: ユーザーの電子メール アドレス
- **`Comment`**: ユーザーに関するコメント
- **`Portrait`**: ユーザーに関するイメージの URL

ユーザーのプロファイルには `Sitecore.Security.Accounts.User.Profile` プロパティを介してアクセスすることができます。たとえば、コンテキスト ユーザーの電子メール アドレスにアクセスする方法は、次のとおりです:

```
Sitecore.Security.Accounts.User user = Sitecore.Context.User;
Sitecore.Security.UserProfile profile = user.Profile;
string userEmail = profile.Email;
```

重要

ユーザー プロファイル プロパティを設定した後は `Sitecore.Security.UserProfile.Save()` メソッドを呼び出す必要があります。次はその例です:

```
Sitecore.Security.Accounts.User user = Sitecore.Context.User;
Sitecore.Security.UserProfile profile = user.Profile;
profile.Email = "address@domain.tld";
profile.Save();
```

メモ

認証されていないユーザーに対してプロファイル プロパティを設定することはできません。認証されたユーザーを読み取るには、`Sitecore.Security.Accounts.User.FromName()` に対する第 2 パラメーターとして True を渡します。たとえば、ドメイン `domain` 内で認証されたユーザー `user` を読み取る方法は、次のとおりです:

```
Sitecore.Security.Accounts.User user =
  Sitecore.Security.Accounts.User.FromName(@"domain\user",true);
```

3.3 カスタム ユーザー プロファイル プロパティへのアクセス方法

`Sitecore.Security.UserProfile` クラスは、カスタム ユーザー プロファイル プロパティの設定、読み取りおよび削除を行うためのメソッドを提供します。

`Sitecore.Security.UserProfile.GetCustomPropertyNames()` メソッドは、ユーザーのプロファイル内のカスタム プロパティの名前を返します。

重要

`Sitecore.Security.UserProfile.GetCustomPropertyNames()` メソッドは、ユーザーに定義されているプロパティ名のみを返します。これは、このソリューションによって使用されるすべてのカスタム プロパティ名の完全なリストではない可能性があります。

`Sitecore.Security.UserProfile.GetCustomProperty()` メソッドは、最初のパラメーターによって指定されたカスタム プロファイル プロパティの値を返します。たとえば、コンテキスト ユーザーに定義されているすべてのカスタム プロパティの名前および値を出力する方法は、次のとおりです:

```
        Sitecore.Security.Accounts.User user = Sitecore.Context.User;
        Sitecore.Security.UserProfile profile = user.Profile;

        foreach(string attributeKey in profile.GetCustomPropertyNames())
        {
          string attributeValue = profile.GetCustomProperty(attributeKey);
          //TODO: handle attributeKey and attributeValue
        }
```

メモ

`Sitecore.Security.UserProfile.GetCustomProperty()` メソッドを使用する代わりに、`Sitecore.Security.UserProfile` クラスによって提供されるコレクションを介してカスタム ユーザー プロファイル プロパティにアクセスすることができます。次はその例です:

```
        Sitecore.Security.Accounts.User user = Sitecore.Context.User;
        Sitecore.Security.UserProfile profile = user.Profile;
        string attributeValue = profile[attributeKey];
```

`Sitecore.Security.UserProfile.SetCustomProperty()` メソッドは、最初のパラメーターによって指定されたカスタム プロファイル プロパティを、2 番目のパラメーターによって指定された値に設定します。認証されていないユーザーに対してカスタム ユーザー プロファイル プロパティを設定することはできません。このメソッドを呼び出した後は `Sitecore.Security.UserProfile.Save()` を呼び出す必要があります。たとえば、コンテキスト ユーザーのプロファイル内で `attributeKey` という名前のカスタム プロパティを値 `attributeValue` に設定する方法は、次のとおりです:

```
        Sitecore.Security.Accounts.User user = Sitecore.Context.User;
        Sitecore.Security.UserProfile profile = user.Profile;
        profile.SetCustomProperty("attributeKey ","attributeValue");
        profile.Save();
```

メモ

`Sitecore.Security.UserProfile.SetCustomProperty()` メソッドを使用する代わりに、`Sitecore.Security.UserProfile` クラスによって提供されるコレクションを介してカスタム プロパティを設定することができます。次はその例です:

```
Sitecore.Security.Accounts.User user = Sitecore.Context.User;
Sitecore.Security.UserProfile profile = user.Profile;
profile["attributeKey"] = "attributeValue";
profile.Save();
```

`Sitecore.Security.UserProfile.RemoveCustomProperty()` メソッドは、ユーザーのプロパティからカスタム プロパティを削除します。認証されていないユーザーからカスタム ユーザー プロファイル プロパティを削除することはできません。このメソッドを呼び出した後は `Sitecore.Security.UserProfile.Save()` を呼び出す必要があります。たとえば、コンテキスト ユーザーのプロファイルから `attributeKey` という名前のカスタム プロパティを削除する方法は、次のとおりです:

```
Sitecore.Security.Accounts.User user = Sitecore.Context.User;
Sitecore.Security.UserProfile profile = user.Profile;
profile.RemoveCustomProperty("attributeKey");
profile.Save();
```

3.4 デフォルト ユーザー プロファイルの拡張方法

デフォルト ユーザー プロファイルを拡張してカスタム ユーザー プロファイル プロパティを追加することができます。デフォルト ユーザー プロファイルを拡張すると、前のセクションの「カスタム ユーザー プロファイル プロパティへのアクセス方法」で説明されている API 以外にもユーザー マネージャーを介してカスタム ユーザー プロファイル プロパティにアクセスすることができます。

デフォルト ユーザー プロファイルを拡張する方法:

1. Sitecore デスクトップで Core データベースを選択します。[16]
2. テンプレート マネージャーまたはコンテンツ エディターで、`/Sitecore/Templates/System/Security/User` データ テンプレート定義アイテムにナビゲートして編集します。
3. 任意のセクションおよびフィールドを追加して新しいデータ テンプレートに対する変更を保存します。
4. Sitecore デスクトップで Master データベースを選択します。

ユーザー マネージャーで、ユーザーをダブルクリックしてから [Profile] タブをクリックし、拡張したプロファイル プロパティにアクセスします。

[16] Sitecore デスクトップでデータベースを選択する方法については、http://sdn.sitecore.net/Reference/Sitecore%206.aspx から「クライアント設定クックブック」を参照してください。

3.5 カスタム ユーザー プロファイルの実装

このセクションでは、カスタム ユーザー プロファイルを実装する手順を示します。

3.5.1 カスタム ユーザー プロファイルの作成方法

カスタム ユーザー プロファイルを作成する方法:

1. Sitecore デスクトップで Core データベースを選択します。[17]
2. テンプレート マネージャーまたはコンテンツ エディターで、`/Sitecore/Templates/System/Security/User` データ テンプレート定義アイテムを複製します。
3. 任意のフィールドを追加して新しいデータ テンプレートに対する変更を保存します。[18]
4. コンテンツ エディターで `/Sitecore/System/Settings/Security/Profiles` を選択します。
5. カスタム ユーザー プロファイル データ テンプレートを使用して新しいユーザー プロファイル定義アイテムを挿入します。ユーザー マネージャーを介して作成された新しいユーザーのデフォルトとしてこのユーザー プロファイルを設定するには、最初にカスタム ユーザー プロファイル定義アイテムを並べ替えます。
6. Sitecore デスクトップで Master データベースを選択します。

重要
ユーザー マネージャーを介して新しいユーザーを作成する場合、[User Profile] フィールドで適切なユーザー プロファイルを選択します。

[17] Sitecore デスクトップでデータベースを選択する方法については、http://sdn.sitecore.net/Reference/References%20in%20Japanese.aspx から『クライアント設定クックブック』を参照してください。
[18] データ テンプレートにフィールドを追加する方法については、http://sdn.sitecore.net/Reference/References%20in%20Japanese.aspx から『データ定義クックブック』を参照してください。

3.5.2 ユーザー マネージャーを使用したカスタム ユーザー プロファイルの適用方法

ユーザー マネージャーを使用してユーザーにカスタム ユーザー プロファイルを適用する方法:

1. ユーザー マネージャーでユーザーを選択します。
2. [ユーザー] グループで [編集] をクリックします。[ユーザーを編集] ダイアログが表示されます。
3. [ユーザーを編集] ダイアログで [プロファイル] タブをクリックします。
4. [変更] をクリックします。[ユーザー プロファイルを変更します] ダイアログが表示されます。
5. [ユーザー プロファイルを変更します] ダイアログで、カスタム ユーザー プロファイルを選択してから [変更] をクリックします。

3.5.3 API を使用したカスタム ユーザー プロファイルの適用方法

`Sitecore.Security.UserProfile.ProfileItemId` プロパティには、Core データベース内のユーザー プロファイル定義アイテムの ID が含まれます。このプロパティを設定した後は `Sitecore.Security.UserProfile.Save()` を呼び出す必要があります。たとえば、コンテキスト ユーザーのカスタム プロファイル定義アイテムを Core データベース内のカスタム ユーザー プロファイル定義アイテム `/sitecore/system/settings/security/profiles/customuserprofile` に設定する方法は、次のとおりです:

```
        string profilePath = "/sitecore/system/settings/security/profiles/customuserprofile";
        Sitecore.Security.Accounts.User user = Sitecore.Context.User;
        Sitecore.Data.Database dbCore = Sitecore.Configuration.Factory.GetDatabase("core");
        Sitecore.Data.Items.Item profileItem = dbCore.GetItem(profilePath);
        user.Profile.ProfileItemId = profileItem.ID.ToString();
        user.Profile.Save();
```

メモ

`Sitecore.Security.UserProfile.ProfileItemId` プロパティには、データ テンプレート自体の ID ではなく、データ テンプレートに基づくアイテムの ID が含まれます。

ヒント

デフォルト プロファイル アイテム ID は、`/App_Config/Security/Domains.Config` 内の各 `/domains/domain` エレメントの `defaultProfileItemId` 属性を使用して指定することができます。

3.5.4 カスタム ユーザー プロファイル クラスの実装方法

カスタム ユーザー プロファイル クラスを実装し、Sitecore.Security.Accounts.User.Profile プロパティによって提供されるデフォルトの Sitecore.Security.UserProfile クラスを置換する方法:

1. Sitecore.Security.UserProfile から継承するカスタム ユーザー プロファイル クラスを作成します。次のコード サンプルを使用することができます:

```
namespace Namespace.Security
{
  public class UserProfile : Sitecore.Security.UserProfile
  {
    public string PropertyName
    {
      get
      {
        return GetCustomProperty("propertyname");
      }
      set
      {
        SetCustomProperty("propertyname", value);
        Save();
      }
    }
  }
}
```

2. web.config 内の /configuration/system.web/profile エレメントの inherits 属性をカスタム ユーザー プロファイル クラスのシグネチャーに更新します:

```
<profile defaultProvider="sql" enabled="true"
  inherits="Namespace.Security.UserProfile,Assembly">
```

3. カスタム ユーザー プロファイル クラスを使用して Sitecore.Security.Accounts.User.Profile プロパティにアクセスします:

```
Namespace.Security.UserProfile profile = Sitecore.Context.User.Profile
  as Namespace.Security.UserProfile;

if(profile!=null)
{
  //TODO: handle profile.PropertyName
}
```

3.6 ユーザー プロファイル管理フォームのサンプル

多くの Web サイトは、ユーザーに関するさまざまなデータが含まれるプロファイルを保持します。パスワードの変更を含むユーザー プロファイルを保持するためのコードビハインドがあるプロファイル管理フォームを収容するサブレイアウトを実装することができます。

メモ
ASP.NET ChangePassword Web コントロールを使用して、ユーザーによるパスワードの変更を許可することもできます。[19] ASP.NET ChangePassword Web コントロールの詳細については、次のセクションの「ASP.NET ChangePassword Web コントロールの使用方法」を参照してください。

メモ
ユーザー プロファイル管理フォーム、あるいはユーザーがプロファイルまたはパスワードを変更できる任意のページに対して未認証アクセス権を付与しないでください。

次のサブレイアウト ファイルのサンプル コードでは、非常に簡単なプロファイル管理データ エントリ フォームを実装しています:

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="Profile.ascx.cs"
  Inherits="Namespace.Web.UI.Profile" %>

Comment: <asp:textbox id="txtComment" runat="server" /><br />
Password: <asp:textbox id="txtPassword" runat="server" textmode="password"/><br />
New Password: <asp:textbox id="txtNewPassword" runat="server" textmode="password"/><br />
Confirm New Password:<asp:textbox id="txtNewPasswordConfirm" runat="server"
  textmode="password"/><br />
<asp:button id="btnGo" text="Go" runat="server" /><br />
<asp:label id="lblMessage" runat="server" />
```

このプロファイル管理フォームには、次が含まれます:

- ユーザーがプロファイルにストアするコメントを入力するためのテキスト フィールド。
- ユーザーが既存のパスワードを入力するためのパスワード フィールド。
- ユーザーが新しいパスワードを入力するためのパスワード フィールド。
- 新しいパスワードを確認するためのパスワード フィールド。これにより、ユーザーがパスワードを誤って入力していないかどうかを確認することができます。

[19] ASP.NET ChangePassword Web コントロールの詳細については、
http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.changepassword.aspx を参照してください。

- フォームをサブミットするためのボタン。
- フォームのサブミットの結果として生じるエラー メッセージを収容するためのラベル。

次のサブレイアウト コードビハインド ファイルのサンプル コードでは、ユーザーのプロファイルを更新するためのロジックを実装しています:

```
using System;

namespace Namespace.Web.UI
{
  public partial class Profile : System.Web.UI.UserControl
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      if(!Sitecore.Context.IsLoggedIn)
      {
        Sitecore.Web.WebUtil.Redirect("/");
      }
      else
      {
        if(IsPostBack)
        {
          lblMessage.Text = "No change to implement.";

          if(String.IsNullOrEmpty(txtPassword.Text))
          {
            lblMessage.Text = "Existing password required.";
          }
          else if(txtNewPassword.Text != txtNewPasswordConfirm.Text)
          {
            lblMessage.Text = "Passwords do not match.";
          }
          else
          {
            Sitecore.Security.Authentication.AuthenticationHelper authHelper =
              new Sitecore.Security.Authentication.AuthenticationHelper(
              Sitecore.Security.Authentication.AuthenticationManager.Provider);

            try
            {
              if(!authHelper.ValidateUser(Sitecore.Context.User.Name, txtPassword.Text))
              {
                throw new System.Security.Authentication.AuthenticationException(
                  "Incorrect password.");
              }
              else
              {
                if(txtComment.Text != Sitecore.Context.User.Profile.Comment)
                {
                  Sitecore.Context.User.Profile.Comment = txtComment.Text;
                  Sitecore.Context.User.Profile.Save();
                  lblMessage.Text = "Comment changed.";
                }

                if((!String.IsNullOrEmpty(txtNewPassword.Text))
                  || !String.IsNullOrEmpty(txtNewPasswordConfirm.Text))
                {
                  System.Web.Security.MembershipUser user
                    = System.Web.Security.Membership.GetUser(
                    Sitecore.Context.User.Name);
```

```
                        if(user.ChangePassword(txtPassword.Text, txtNewPassword.Text))
                        {
                          lblMessage.Text = "Password changed.";
                        }
                        else
                        {
                          throw new System.Security.Authentication.AuthenticationException(
                            "Unable to change password");
                        }
                      }
                    }
                  }
                  catch(System.Security.Authentication.AuthenticationException)
                  {
                    lblMessage.Text = "Processing error.";
                  }
                }
              }
              else
              {
                txtComment.Text = Sitecore.Context.User.Profile.Comment;
              }
            }
          }
        }
      }
```

このコードの背後にあるロジックは、次のとおりです:

1. ユーザーが認証されていない場合、ホーム ページにリダイレクトし、それ以上の処理は行いません。ユーザーは、プロファイルを更新する前に認証される必要があります。

2. ページがポストバックしていない場合、ユーザーにはデータをフォームに入力する機会がありません。この場合、コメント フィールドにユーザーのプロファイルからコメントを投入し、それ以上の処理は行いません。

3. エラー メッセージを、プロファイル属性が変更されていないことを示すデフォルト値に更新します。

4. ユーザーによってパスワードが入力されていない場合、エラー メッセージを表示し、それ以上の処理は行いません。

5. ユーザーによって新しいパスワード フィールドと新しいパスワードの確認フィールドのどちらかに新しいパスワードが入力され、これらのフィールドの値が一致しない場合、エラー メッセージを表示し、それ以上の処理は行いません。

6. ユーザーによって入力されたパスワードがコンテキスト ユーザーに対して無効である場合、例外をスローします。これにより、汎用エラー メッセージを表示し、それ以上の処理は行いません。

7. ユーザーによって入力されたコメントがユーザーのプロファイルにストアされているコンテンツと異なる場合、ユーザーのプロファイル内のコメントを更新し、コメントが変更されたことを示すメッセージを表示します。

8. ユーザーによって新しいパスワードが入力されたときに、ユーザーに関連付けられたパスワードを更新できる場合、パスワードが変更されたことを示すメッセージを表示し、それ以上の処理は行いません。

9. ユーザーに関連付けられたパスワードを変更できなかった場合、例外をスローします。これにより、汎用エラー メッセージを表示し、それ以上の処理は行いません。

メモ
プロファイル管理ページにアクセスするための認証が必要な場合、ユーザーが自己登録する場合は特に、ユーザーがプロファイルを更新する際にパスワードを求める必要は必要ありません。セキュリティ上、ユーザーがフォームにアクセスしてから一定期間が経過してもプロファイルの変更をサブミットしない場合はパスワードを求める必要があります。別のユーザーがブラウザーにアクセスする前にユーザーがログオフし忘れた際に他のユーザーによるユーザーのプロファイルの更新を阻止するには、パスワードが必要です。ユーザーのパスワードを更新する場合は特にパスワードが必要です。

3.6.1 ASP.NET ChangePassword Web コントロールの使用方法

コードビハインドを作成する代わりに ASP.NET ChangePassword Web コントロールを使用してユーザーによるパスワードの変更を許可することができます。[20]

たとえば、ASP.NET ChangePassword Web コントロールをサブレイアウトに追加します:

```
        <asp:changepassword id="changePasswordControl" runat="server" />
```

ASP.NET Login Web コントロールは、カスタム コードビハインドなしでコンテキスト ユーザーのパスワードを変更します。

[20] ASP.NET ChangePassword Web コントロールの詳細については、
http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.changepassword.aspx を参照してください。

Chapter 4

アクセス権管理

この章では、ソリューションについて説明し、アクセス権を管理するためのサンプル コードを示します。この章では、アクセス権の概要、アクセス権を無効にする 2 つのテクニック、アイテムのアクセス権を更新するコードの例を示します。

この章には次のセクションがあります:

- アクセス権の概要
- ユーザー スイッチャー
- セキュリティ ディセーブラー
- アクセス権の適用

4.1 アクセス権の概要

アクセス権は、フィールド値の読み取りおよび書き込みや子アイテムの挿入を含むアイテム内のデータに対するさまざまな操作を実行できるユーザーおよびロールを制御するものです。コンテキスト ユーザーが読み取りアクセス権を所有していないアイテムにコードがアクセスすると、システムはこのアイテムが存在しないかのように動作します。コンテキスト ユーザーが読み取りアクセス権を所有しているが書き込みアクセス権は所有していないアイテムをコードが更新しようとすると、例外がスローされます。コンテキスト ユーザーが読み取りアクセス権を所有しているが挿入アクセス権は所有していないアイテムの下にコードがアイテムを作成しようとすると、例外がスローされます。

場合によっては、コンテキスト ユーザーが特定のタスクを実行するためのアクセス権を所有していなくても、コードのブロックがこのタスクを実行できるようにすることができます。この場合、次のセクションの「ユーザー スイッチャー」で説明されているように、ユーザー スイッチャーを使用して、特定のユーザーのコンテキストでコードのセグメントを実行することができます。また、次のセクションの「セキュリティ ディセーブラー」で説明されているように、セキュリティ ディセーブラーを使用して、管理権限を持つユーザーのコンテキストでコードのセグメントを実行することもできます。

また、次のセクションの「アクセス権の適用」で説明されているように、アイテムおよび場合によってはその子孫に関連付けられたアクセス権を更新することもできます。

4.2 ユーザー スイッチャー

`Sitecore.Security.Accounts.UserSwitcher` クラスを使用して、コンテキスト ユーザーとは関係なく、特定のユーザーのコンテキストでコードのセグメントを実行することができます。このアプローチを使用するには、`Sitecore.Security.Accounts.UserSwitcher` をリソースとして C# `using` 文に渡します。[21]

`Sitecore.Security.Accounts.UserSwitcher` コンストラクターは、コンテキスト ユーザーを指定したユーザーに設定します。`using` 文ブロック内のコードには、`Sitecore.Security.Accounts.UserSwitcher` クラスのコンストラクターに渡された最初のパラメーターによって指定されたユーザーの有効権限があります。ユーザーにロールを割り当て、アイテムにアクセス権を適用することにより、ユーザー スイッチャー内のコードのブロックが特定のアイテムに対して実行可能な操作を正確に制御することができます。ブロックが終了すると、`using` 文により、NET ランタイム エンジンが `Sitecore.Security.Accounts.UserSwitcher.Dispose()` を呼び出し、コンテキスト ユーザーを元のコンテキスト ユーザーにリセットします。

たとえば、ドメイン `domain` 内のユーザー `user` としてコードのセグメントを呼び出す方法は、次のとおりです:

```
        string domainUser = @"domain\user";

        if(Sitecore.Security.Accounts.User.Exists(domainUser))
        {
          Sitecore.Security.Accounts.User user =
            Sitecore.Security.Accounts.User.FromName(domainUser,false);

          using(new Sitecore.Security.Accounts.UserSwitcher(user))
          {
            //TODO: code to invoke as user
          }
        }
```

メモ

アイテムを更新するには、`Sitecore.Data.Items.Item.Editing.BeginEdit()` を使用して最初にアイテムを編集状態にする必要があります。その後、`Sitecore.Data.Items.Item.Editing.EndEdit()` または `Sitecore.Data.Items.Item.Editing.CancelEdit()` を使用してトランザクションをコミットまたはロールバックします。たとえば、コンテキスト アイテムを更新する方法は、次のとおりです:

```
        string domainUser = @"domain\user";

        if(Sitecore.Security.Accounts.User.Exists(domainUser))
        {
          Sitecore.Security.Accounts.User user =
            Sitecore.Security.Accounts.User.FromName(domainUser,true);

          using(new Sitecore.Security.Accounts.UserSwitcher(user))
          {
            Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
            contextItem.Editing.BeginEdit();
```

[21] C# `using` 文の詳細については、http://msdn.microsoft.com/en-us/library/yh598w02.aspx を参照してください。

```
            try
            {
              //TODO: update contextItem
              contextItem.Editing.EndEdit();
            }
            catch(Exception ex)
            {
              contextItem.Editing.CancelEdit();
            }
          }
        }
```

4.3 セキュリティ ディセーブラー

`Sitecore.SecurityModel.SecurityDisabler` クラスを使用して、コンテキスト ユーザーとは関係なく、管理権限を持つユーザーのコンテキストでコードのセグメントを実行することができます。このアプローチを使用するには、`Sitecore.SecurityModel.SecurityDisabler` をリソースとして C# `using` 文に渡します。`using` 文ブロック内のコードには、システム全体を完全に制御でき、任意のアイテムまたはフィールドに対して任意のアクションを実行することができます。

たとえば、管理権限を持つセキュリティ コンテキスト内でコードのセグメントを呼び出す方法は、次のとおりです:

```
using(new Sitecore.SecurityModel.SecurityDisabler())
{
  //TODO: code to invoke as administrator
}
```

メモ

アイテムを更新するには、`Sitecore.Data.Items.Item.Editing.BeginEdit()` を使用して最初にアイテムを編集状態にする必要があります。その後、`Sitecore.Data.Items.Item.Editing.EndEdit()` または `Sitecore.Data.Items.Item.Editing.CancelEdit()` を使用してトランザクションをコミットまたはロールバックします。たとえば、コンテキスト アイテムを更新する方法は、次のとおりです:

```
using(new Sitecore.SecurityModel.SecurityDisabler())
{
  Sitecore.Data.Items.Item contextItem = Sitecore.Context.Item;
  contextItem.Editing.BeginEdit();

  try
  {
    //TODO: update contextItem
    contextItem.Editing.EndEdit();
  }
  catch(Exception ex)
  {
    contextItem.Editing.CancelEdit();
  }
}
```

4.4 アクセス権の適用

次のようなコードを使用して、Sitecore データベース内のアイテムにアクセスを設定することができます。

```
        private void SetRight(Sitecore.Data.Items.Item item,
          Sitecore.Security.Accounts.Account account,
          Sitecore.Security.AccessControl.AccessRight right,
          Sitecore.Security.AccessControl.AccessPermission rightState,
          Sitecore.Security.AccessControl.PropagationType propagationType)
        {
          Sitecore.Security.AccessControl.AccessRuleCollection accessRules =
            item.Security.GetAccessRules();

          if(propagationType == Sitecore.Security.AccessControl.PropagationType.Any)
          {
            accessRules.Helper.RemoveExactMatches(account, right);
          }
          else
          {
            accessRules.Helper.RemoveExactMatches(account, right, propagationType);
          }

          if(rightState != Sitecore.Security.AccessControl.AccessPermission.NotSet)
          {
            if(propagationType == Sitecore.Security.AccessControl.PropagationType.Any)
            {
              accessRules.Helper.AddAccessPermission(account, right,
                Sitecore.Security.AccessControl.PropagationType.Entity, rightState);
              accessRules.Helper.AddAccessPermission(account, right,
                Sitecore.Security.AccessControl.PropagationType.Descendants, rightState);
            }
            else
            {
              accessRules.Helper.AddAccessPermission(account, right, propagationType,
                rightState);
            }
          }

          item.Security.SetAccessRules(accessRules);
        }

        private void SetRight(string strDatabase, string strItem, string strAccount,
          string strRight, Sitecore.Security.AccessControl.AccessPermission rightState,
          Sitecore.Security.AccessControl.PropagationType propagationType)
        {
          Sitecore.Data.Database db = Sitecore.Configuration.Factory.GetDatabase(strDatabase);
          Sitecore.Data.Items.Item item = db.GetItem(strItem);
          Sitecore.Security.Accounts.AccountType accountType =
            Sitecore.Security.Accounts.AccountType.User;

          if(Sitecore.Security.SecurityUtility.IsRole(strAccount))
          {
            accountType = Sitecore.Security.Accounts.AccountType.Role;
          }

          Sitecore.Security.Accounts.Account account =
            Sitecore.Security.Accounts.Account.FromName(strAccount, accountType);
          Sitecore.Security.AccessControl.AccessRight right =
            Sitecore.Security.AccessControl.AccessRight.FromName(strRight);
          SetRight(item, account, right, rightState, propagationType);
        }
```

このサンプル コードには、2 つのメソッドが含まれます。最初のメソッドは、オブジェクト パラメーターを受け入れます。2 番目のメソッドは、文字列パラメーターを受け入れ、これらをオブジェクトに変換し、最初のメソッドを呼び出します。最初のメソッドの背後にあるロジックは、次のとおりです:

1. アイテムのアクセス ルールを読み取ります。

2. 呼び出し元から `PropagationType` として `Any` が指定された場合、アクセス ルールの変更がアイテムとその子孫に適用されます。この場合、指定されたアイテムとその子孫から指定されたアカウントに対するアクセス権を削除します。それ以外の場合、指定されたアイテム、および場合によっては指定された伝播の種類に応じてその子孫から、指定されたアカウントに対するアクセス権を削除します。

3. 呼び出し元から `AccessPermission` として `NotSet` が指定された場合、アカウントには新しいアクセス権が適用されません。それ以外の場合、呼び出し元から `PropagationType` として `Any` が指定された場合、指定されたアカウントに対して指定したアクセス権をアイテムとその子孫に適用します。呼び出し元から他の任意の `PropagationType` が指定された場合、この値に基づいてアクセス権を適用します。

4. アクセス権の変更をアイテムにコミットします。

重要
このコードではロールは説明されていません。ユーザーのアクセス権を削除してもその任意のロールのアクセス権は削除されず、ロールのアクセス権を削除してもネストされたロールには影響しません。

Chapter 5

System.Web.Security API

この章では、Sitecore API によって提供されない共通セキュリティ操作を実装するための .NET API について説明します。

この章には次のセクションがあります:

- System.Web.Security.Roles
- System.Web.Security.MembershipUser
- System.Web.Security.Membership

5.1 System.Web.Security.Roles

`System.Web.Security.Roles` クラスは、次の各セクションで説明されているように、Sitecore セキュリティ API によって抽象化されないセキュリティ機能を提供します。

5.1.1 System.Web.Security.Roles.CreateRole()

`System.Web.Security.Roles.CreateRole()` メソッドは、ドメイン内にロールを作成します。たとえば、ロール `role` が存在しないときにこのロールをドメイン `domain` 内に作成する方法は、次のとおりです:

```
        string domainRole = @"domain\role";

        if(!Sitecore.Security.Accounts.Role.Exists(domainRole))
        {
          System.Web.Security.Roles.CreateRole(domainRole);
        }
```

5.1.2 System.Web.Security.Roles.DeleteRole()

`System.Web.Security.Roles.DeleteRole()` メソッドは、最初のパラメーターによって指定されたロールからすべてのメンバーを削除してから、このロールを削除します。たとえば、ドメイン `domain` からロール `role` を削除する方法は、次のとおりです:

```
        string domainRole = @"domain\role";

        if(Sitecore.Security.Accounts.Role.Exists(domainRole))
        {
          System.Web.Security.Roles.DeleteRole(domainRole);
        }
```

メモ
ユーザーの数によっては、ロールの削除には長時間かかる場合があります。

5.2 System.Web.Security.MembershipUser

元になる .NET メンバーシップ プロバイダーは、System.Web.Security.MembershipUser クラスを使用してユーザーを表します。次の各セクションでは、Sitecore セキュリティ API によって提供されないこのクラスで使用可能な機能について説明します。

5.2.1 System.Web.Security.MembershipUser.GetUser()

System.Web.Security.MembershipUser.GetUser() メソッドは、最初のパラメーターによって指定された System.Web.Security.MembershipUser を返します。たとえば、System.Web.Security.MembershipUser としてコンテキスト ユーザーにアクセスする方法は、次のとおりです:

```
        Sitecore.Security.Accounts.User user = Sitecore.Context.User;
        System.Web.Security.MembershipUser mUser =
          System.Web.Security.Membership.GetUser(user.Name);
```

5.2.2 System.Web.Security.MembershipUser.ChangePassword()

System.Web.Security.MembershipUser.ChangePassword() メソッドは、ユーザーのパスワードを変更します。ユーザーのパスワードを変更できない場合は、False を返します。たとえば、現在のパスワードが oldPassword である際にコンテキスト ユーザーのパスワードを newPassword に設定する方法は、次のとおりです:

```
        Sitecore.Security.Accounts.User user = Sitecore.Context.User;
        System.Web.Security.MembershipUser mUser =
          System.Web.Security.Membership.GetUser(user.Name);

        if (!mUser.ChangePassword("oldPassword","newPassword"))
        {
          //TODO: handle case that password was not changed
        }
```

5.2.3 System.Web.Security.MembershipUser.ChangePasswordQuestionAndAnswer()

System.Web.Security.MembershipUser.ChangePasswordQuestionAndAnswer() メソッドは、ユーザーのパスワード、セキュリティ上の質問および回答を変更します。これらを変更できない場合は、False を返します。たとえば、コンテキスト ユーザーのパスワードを newPassword、セキュリティ上の質問を newQuestion、その回答を newAnswer に設定する方法は、次のとおりです:

```
        Sitecore.Security.Accounts.User user = Sitecore.Context.User;
        System.Web.Security.MembershipUser mUser =
          System.Web.Security.Membership.GetUser(user.Name);

        if (!mUser.ChangePasswordQuestionAndAnswer("newPassword","newQuestion","newAnswer"))
        {
```

```
      //TODO: handle case that password was not changed
    }
```

5.2.4 System.Web.Security.MembershipUser.ResetPassword()

`System.Web.Security.MembershipUser.ResetPassword()` メソッドは、ユーザーのパスワードをランダム文字列に変更し、この文字列を返します。たとえば、コンテキスト ユーザーのパスワードをランダム化する方法は、次のとおりです:

```
    Sitecore.Security.Accounts.User user = Sitecore.Context.User;
    System.Web.Security.MembershipUser mUser =
      System.Web.Security.Membership.GetUser(user.Name);
    string password = mUser.ResetPassword();
```

`System.Web.Security.MembershipUser.ResetPassword()` メソッドは、ユーザーのセキュリティ パスワードに対する回答を求めるシステムに追加シグネチャーを提供します。たとえば、セキュリティ上の質問に対する回答が `answer` である際にコンテキスト ユーザーのパスワードをランダム化する方法は、次のとおりです:

```
    Sitecore.Security.Accounts.User user = Sitecore.Context.User;
    System.Web.Security.MembershipUser mUser =
      System.Web.Security.Membership.GetUser(user.Name);
    string password = mUser.ResetPassword("answer");
```

5.2.5 System.Web.Security.MembershipUser.UnlockUser()

`System.Web.Security.MembershipUser.UnlockUser()` メソッドは、許容可能な回数を超えて無効なパスワードを入力したためにロックアウトされているユーザーのロックを解除します。ユーザーのロックを解除できない場合、このメソッドは False を返し、それ以外の場合は True を返します。たとえば、ドメイン `domain` 内のユーザー `user` のロックを解除する方法は、次のとおりです:

```
    string domainUser = @"domain\user";

    if(Sitecore.Security.Accounts.User.Exists(domainUser))
    {
      System.Web.Security.MembershipUser mUser =
        System.Web.Security.Membership.GetUser(domainUser);

      if(!mUser.UnlockUser())
      {
        //TODO: handle case that system is not able to unlock user
      }
    }
```

ユーザーがロックアウトされる前に無効なパスワードを入力できる回数の設定の詳細については、「メンバーシップ プロバイダー設定」のセクションを参照してください。

5.3 System.Web.Security.Membership

次の各セクションでは、Sitecore セキュリティ API によって提供されない `System.Web.Security.Membership` クラスで使用可能な機能について説明します。

5.3.1 System.Web.Security.Membership.GetUserNameByEmail()

対応するユーザーが 1 人のみであることを前提として、`System.Web.Security.Membership.GetUserNameByEmail()` メソッドは、最初のパラメーターによって指定された電子メール アドレスに関連付けられたユーザーの名前を読み取ります。たとえば、電子メール アドレス `Sitecore.Security.Acconts.User` に関連付けられた `address@domain.tld` を処理する方法は、次のとおりです:

```
string domainUser =
  System.Web.Security.Membership.GetUserNameByEmail("address@domain.tld");

if((!String.IsNullOrEmpty(domainUser))
  && Sitecore.Security.Accounts.User.Exists(domainUser))
{
  Sitecore.Security.Accounts.User user =
    Sitecore.Security.Accounts.User.FromName(domainUser,false);
  //TODO: handle user
}
```

5.3.2 System.Web.Security.Membership.FindUsersByEmail()

`System.Web.Security.Membership.FindUsersByEmail()` メソッドは、最初のパラメーターによって指定された電子メール アドレスに関連付けられた `System.Web.Security.MembershipUser` オブジェクトのリストを返します。たとえば、電子メール アドレス `address@domain.tld` に関連付けられた `Sitecore.Security.Accounts.User` を処理する方法は、次のとおりです:

```
foreach(System.Web.Security.MembershipUser mUser in
  System.Web.Security.Membership.FindUsersByEmail("address@domain.tld"))
{
  Sitecore.Security.Accounts.User user =
    Sitecore.Security.Accounts.User.FromName(mUser.UserName,false);
  //TODO: handle user
}
```

Chapter 6

付録 A

この付録では、記号が API 内のエレメントにどのようにマッピングされるかを示します。

この付録には次のセクションがあります:

- Sitecore.Security.AccessControl.AccessRight

6.1 Sitecore.Security.AccessControl.AccessRight

`Sitecore.Security.AccessControl.AccessRight` クラスは、`web.config` 内の `/configuration/sitecore/accessRights` に定義されている個々のアクセス権を表します。
`Sitecore.Security.AccessControl.AccessRight` クラスの次のプロパティは、さまざまなアクセス権を表します。

プロパティ	アクセス権コード
Any	*
FieldRead	field:read
FieldWrite	field:write
InsertShow	insert:show
ItemAdmin	item:admin
ItemCreate	item:create
ItemDelete	item:delete
ItemRead	item:read
ItemRename	item:rename
ItemWrite	item:write
LanguageRead	language:read
LanguageWrite	language:write
SiteEnter	site:enter
WorkflowCommandExecute	workflowCommand:execute
WorkflowStateDelete	workflowState:delete
WorkflowStateWrite	workflowState:write