ASP.NET 4.0 in Practice

Daniele Bochicchio
Stefano Mostarda
Marco De Sanctis

Includes 106 practical techniques

MANNING

SAMPLE CHAPTER

ASP.NET 4.0 in Practice

by Daniele Bochicchio,
Stefano Mostarda,
and Marco De Sanctis

Chapter 15

brief contents

PART 1 ASP.NET FUNDAMENTALS 1

1 ■ Getting acquainted with ASP.NET 4.0 3
2 ■ Data access reloaded: Entity Framework 30
3 ■ Integrating Entity Framework and ASP.NET 52

PART 2 ASP.NET WEB FORMS 75

4 ■ Building the user interface with ASP.NET Web Forms 77
5 ■ Data binding in ASP.NET Web Forms 104
6 ■ Custom controls 135
7 ■ Taking control of markup 162

PART 3 ASP.NET MVC 185

8 ■ Introducing ASP.NET MVC 187
9 ■ Customizing and extending ASP.NET MVC 219

PART 4 SECURITY 257

10 ■ ASP.NET security 259
11 ■ ASP.NET authentication and authorization 282

PART 5 ADVANCED TOPICS.....317

12 ■ Ajax and RIAs with ASP.NET 4.0 319
13 ■ State 348
14 ■ Caching in ASP.NET 366
15 ■ Extreme ASP.NET 4.0 396
16 ■ Performance and optimizations 416

15 Extreme ASP.NET 4.0

This chapter covers

- `HttpModules`
- Logging and error handling
- Extending `HttpRuntime`
- How to build a virtual path provider

Extensibility is a driving force of ASP.NET, and this chapter is composed of different techniques used to implement advanced—and probably extreme—ASP.NET-based features.

We described `HttpModules` and `HttpHandlers` in chapter 1 from an architectural point of view. In this chapter, we'll use them to implement common strategies in websites; for example, we'll look at error handling, which is fundamental for both security and troubleshooting. We'll use multithreading techniques to increase performance in specific scenarios. Finally, we'll talk about how `HttpRuntime` extensibility can address your remaining needs, letting you store your own source in any nonconventional storage, such as a database or even remote servers.

This chapter and the next are the last in the book, and we've already covered everything you'll see from now on, to some degree, in the previous chapters. This chapter, in particular, is organized to show you advanced topics related to

HttpRuntime and multithreading. If you need out-of-the-ordinary techniques in your application, this chapter is for you.

15.1 Using HttpModules

As we mentioned in chapter 1, HttpModules are a special kind of class used to intercept mainly HttpApplication events (but you can handle events from any object if you need to). An HttpModule implements the IHttpModule interface from the System.Web namespace and is loaded at runtime. Generally, HttpModules are stateless with regard to the current request, so they don't contain state related to the current request, but they do use HttpContext (a singleton class) as their state context.

HttpContext offers access to both HttpRequest and HttpResponse, enabling state to be used across request and response. You also have the ability to use session state, caching, and application state.

Each HttpApplication has only one instance of a given HttpModule. Remember that you can have different instances of HttpApplication in a given web application, depending on the ASP.NET HttpApplication pool configuration (not to be confused with IIS ones), or in case ASP.NET demands more. (For a complete rundown of the details of this topic, see chapter 1.) This single-instance behavior is reflected by IHttpModule interface members, which are composed of a simple Init() member, used to initialize elements, and a Dispose() member, optionally used to clean up resources if you need to do that.

Migrating HttpHandlers and HttpModules to the IIS 7.0 integrated pipeline

To enable HttpHandlers and HttpModules in the IIS 7.0 integrated pipeline, you need to move the data under the system.WebServer node, instead of under system.web. You can automatically do this with the following command-line tool:

```
%windir%\system32\inetsrv\APPCMD.EXE migrate config <Application Path>
```

To avoid a runtime error when the legacy httpModules section is present (for example, if you need to deploy this application to both IIS 6.0/7.0 in classic pipeline and IIS 7.0 in integrated pipeline), you can set validateIntegratedModeConfiguration under system.webServer\validation.

You can also use a shortcut to enable all managed modules to run for all requests in your application, regardless of the preCondition attribute (to be set to managedHandler), by setting the runAllManagedModulesForAllRequests property in the system.webServer\modules section.

To build an HttpModule, you need to register it in the web.config file. Depending on your IIS version, you can make an HttpModule globally available and use it across all kinds of requests. For information about this specific feature, available on IIS 7.0 and 7.5, see chapter 1.

> **NOTE** `HttpApplication` has different events, giving you full control over which ASP.NET state you need to capture, either request or response. You can find all the events in the documentation, which is also available on MSDN at http://www.mng.bz/SeWM.

`HttpModules` are considered the heart of ASP.NET because common features are implemented with it: `OutputCache`, `SessionState`, authorization, and authentication, to name a few. Extensibility in ASP.NET often depends on `HttpModules` because they enable you to modify virtually anything related to the response and request flows. This section is dedicated to leveraging `HttpApplication`.

TECHNIQUE 90 Modifying the response flow with HttpModules

`HttpModules` can modify every single aspect of ASP.NET, so you can use them to manipulate the response flow before you send the output straight to the browser. This technique can be useful in a lot of scenarios: you can add specific headers to specific kinds of content or simply modify the flow and redirect the user to a specific page. When you use `HttpModules` creatively, you can deeply influence the way ASP.NET handles the response flow, as we'll show you in this example.

PROBLEM

We want to write a module to handle a custom authorization mechanism for our application. We want to provide a new authorization feature, with our custom logic inside. ASP.NET includes `UrlAuthorizationModule` by default, which is useful for mapping access, via web.config, to a given set of URLs. This custom module will let you dynamically specify authorization rules, so you don't have to rely on static specification with the web.config rules.

SOLUTION

Generally, `BeginRequest` or `EndRequest` events of `HttpApplication` are used the most because you usually need to modify the output either before the corresponding `HttpHandler` begins its work or right after the output is ready to be sent.

The `AuthorizeRequest` and `AuthenticateRequest` events are also useful. They're respectively related to authorization and authentication requests from ASP.NET. They both enable you to customize those mechanisms, as outlined in figure 15.1.

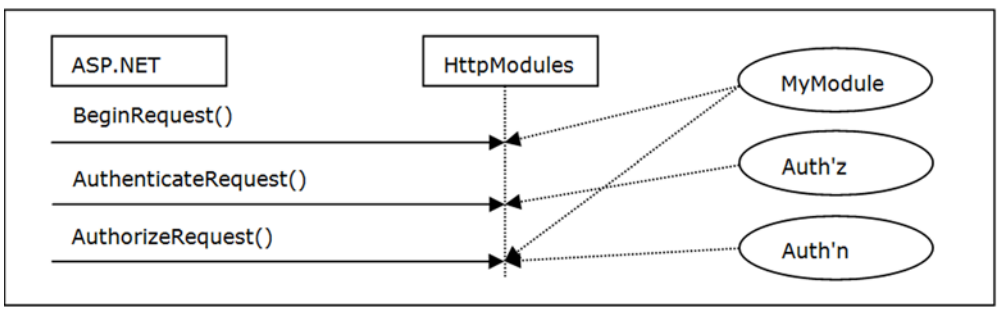

Figure 15.1 The ASP.NET request flow in action. `HttpApplication` events are intercepted by custom modules, so the flow can be changed. In this figure, `MyModule` is a custom module that will intercept `BeginRequest` and `AuthorizeRequest` events.

These events are strictly synchronous, but you can also use their asynchronous equivalents in a fire-and-forget way. Using them asynchronously is handy when you have to deal with data loading or intensive processing routines, where you don't need to modify the request or response status.

For our specific problem, we need to intercept and handle the `AuthorizeRequest` event of the `HttpApplication` class. This event occurs after `BeginRequest` and `AuthenticateRequest` to ensure that the request will be authorized before any handler or module is processed any further.

For our simple example, we're going to intercept the event, and, if the current time is after 5 PM, we'll set the `StatusCode` property of `HttpResponse` to 401, which means that the request isn't authorized. The result is that ASP.NET will stop the request, and, depending on the authentication configuration, the user will be redirected to the login page; in the case of Windows Authentication, the user will be asked for a valid account.

Obviously, you can use a better-fitting dynamic routine, but this solution is a good way for you to get the point regarding authorization customization. The code in the following listing shows how to achieve the result.

Listing 15.1 A custom authorization module to modify the response flow

C#:

```
public class AuthorizationModule : IHttpModule
{
  ...
  public void Init(HttpApplication context)
  {
    context.AuthorizeRequest += new EventHandler(OnAuthorizeRequest);
  }

  void OnAuthorizeRequest (object sender, EventArgs e)
  {
    HttpApplication app = (HttpApplication)sender;

    if (DateTime.Now.Hour >= 17
      app.Context.Response.StatusCode = 401;
  }
}
```

VB:

```
Public Class AuthorizationModule
    Implements IHttpModule
  ...
  Public Sub Init(ByVal context As HttpApplication)
➥       Implements IHttpModule.Init
    AddHandler context.AuthorizeRequest, AddressOf OnAuthorizeRequest
  End Sub

  Private Sub OnAuthorizeRequest(ByVal sender As Object,
                                 ByVal e As EventArgs)
    Dim app As HttpApplication = DirectCast(sender, HttpApplication)

    If DateTime.Now.Hour >= 17 Then
      app.Context.Response.StatusCode = 401
```

```
    End If
  End Sub

End Class
```

The code is self-explanatory: we're intercepting the authorization request and changing the request flow by setting a specific HTTP response code.

As we already mentioned, the `HttpModule` needs to be registered in web.config. It will work on every request coming to ASP.NET (not only those coming to .aspx pages), so if you have special content, like images or style sheet, you should exclude them from its range.

DISCUSSION

Customizing the ASP.NET response flow isn't so difficult: you have to intercept and handle `HttpApplication` events and provide your own custom code in response. This approach could lead to some interesting personalization, using a clean and centralized solution.

Even though the code presented in this example is simple, you can add your own rules to validate the current request authentication, and consequently authorize the response based on your needs. You can get even more creative in your use of `HttpModules`, as you'll see in the next example in which we'll intercept and handle a mobile-device-specific skin.

TECHNIQUE 91 Intercepting and handling mobile device requests

Mobile devices are extremely popular today, but they require a special kind of UI. They have smaller screens, less power, and different screen resolutions than other devices. They can also have different screen orientations: square screen, portrait, or landscape. They need special treatment to use a website to its maximum potential. This example addresses this problem with a solution applied in the heart of the application.

PROBLEM

We want to write a custom action to intercept and manage requests coming from mobile devices. We're going to apply a specific master page because we don't want to let non-mobile users navigate in our specific low-band version; we want to reserve it for the exclusive use of mobile users.

SOLUTION

The solution is simple and is based partially on browser capabilities (see section 7.2). The magic behind this script is in how ASP.NET intercepts and handles our request. The result will be similar to the screenshot shown in figure 15.2.

Figure 15.2 The website, as it will be displayed in its mobile-specific layout. By using a specific version for specific devices, you'll achieve better usability.

Let's suppose that we have the browser definitions updated (or a custom provider in place); all we need to do is check the `IsMobileDevice` property of the `HttpBrowserCapabilities` instance, which we can access through `HttpRequest`.

To indicate that a mobile version is running, we're injecting a special value into `HttpContext.Items` so that we can access it later in our controls. Listing 15.2 contains the code that will help us identify mobile requests and produce the corresponding output in the inner components (for example, it will change the page size for lists or simply provide less content for some specific views).

Listing 15.2 A custom HttpModule to handle mobile devices

C#:

```
public void Init(HttpApplication context)
{
  context.PreRequestHandlerExecute += new EventHandler(CheckMobileRequest);
}

void CheckMobileRequest(object sender, EventArgs e)
{
  HttpApplication app = sender as HttpApplication;
  if (app.Request.Browser.IsMobileDevice)
  {
    app.Context.Items["isMobile"] = true;
    ModifyMasterPage(app);
  }
}

private void ModifyMasterPage(HttpApplication app)
{
  if (app.Context.Handler is Page)
  {
    ((Page)app.Context.Handler).PreInit +=
                      new EventHandler(ApplyMasterPage);
  }
}

private void ApplyMasterPage(object sender, EventArgs e)
{
  ((Page)sender).MasterPageFile = "~/Masters/Mobile.master";
}
```

For mobiles only

Check for page request

Change to mobile

VB:

```
Public Sub Init(ByVal context As HttpApplication) Implements IHttpModule.Init
  AddHandler context.PreRequestHandlerExecute, AddressOf CheckMobileRequest
End Sub

Private Sub CheckMobileRequest(ByVal sender As Object,
                               ByVal e As EventArgs)
  Dim app As HttpApplication = TryCast(sender, HttpApplication)
  If app.Request.Browser.IsMobileDevice Then
    app.Context.Items("isMobile") = True
    ModifyMasterPage(app)
  End If
End Sub

Private Sub ModifyMasterPage(ByVal app As HttpApplication)
  If TypeOf app.Context.Handler Is Page Then
```

For mobiles only

Check for page request

```
    AddHandler DirectCast(app.Context.Handler, Page).PreInit,
               AddressOf ApplyMasterPage
  End If
End Sub

Private Sub ApplyMasterPage(ByVal sender As Object, ByVal e As EventArgs)
  DirectCast(sender, Page).MasterPageFile =
                          "~/Masters/Mobile.master"          <-- Change to mobile
End Sub
```

The code is simple and consists of a series of checks that ensure that we're modifying only requests directed to pages and only those made by mobile devices. The result will look similar to figure 15.2.

DISCUSSION

Instead of a classic example based on `HttpApplication` events, this one is the best way to demonstrate the potential offered by `HttpModules`: you can change every single aspect of the pipeline and plug your code where it's functional for your needs.

The important technique shown in this example is the ability to, from a central point, add an event handler for every page requested. You can do this by using the `Handler` property of `HttpContext`, which contains the current handler assigned to the response. The remaining code is self-explanatory: we changed the master page to `Mobile.master`, which is how the magic of changing the layout occurs.

We created three master pages: one to act as a master for the others (`Main.master`), one for the normal version (`Full.master`), and one for the mobile one (`Mobile.master`). Pages will reference only `Full.master`, which is based on `Main.master` (nested master pages are fully supported); this module will change—on the fly—the value of the `Page.MasterPageFile` property to the corresponding path for our mobile-enabled master page.

The result is truly amazing because it clearly shows you the potential of `HttpModules`. You might arrive at a similar result using a base common class for pages, but the approach we've described here is more versatile because you can apply your own rules to existing applications, or you can apply them granularly to a specific set of pages. All this is possible because you can easily plug `HttpModules` into the pipeline.

The next part is dedicated to another important topic: how to deal with errors, log them, and handle them properly.

15.2 Logging and handling errors

Logging exceptions is important for controlling your applications when they're deployed. You can opt for your own way of storing this information, using a variation of the code shown in listing 15.1 and intercepting the `Error` event of `HttpApplication`, or by using one of the libraries available on the market. Both solutions have their own pros and cons: writing your own code is probably a win/win situation if you don't want to include references to gigantic libraries in order to use only a small portion of their features; using third-part code lets you implement the task in less time.

No matter which method you choose, handling errors the right way is crucial from a security point of view: the less your attacker can see, the more secure your application

is. In this section, you'll learn how to protect your errors from others' eyes, and, at the same time, log them for tracing purposes.

15.2.1 Error logging with Enterprise Library and log4net

If you decide to use custom libraries to handle logs, you'll probably choose between Microsoft Enterprise Library and Apache Software Foundation's (ASF) log4net.

Microsoft Enterprise Library (at the time we were writing this book) is available in version 5.0 at http://www.mng.bz/T85o. This library is free and contains a lot of functionalities, of which logging is only a small part. It's widely used among enterprise applications; even though it's not part of the .NET Framework BCL, developers tend to trust external class libraries that come from Microsoft. log4net is a project from Apache Software Foundation and is available under the Apache License at http://www.mng.bz/0OX6. Both libraries provide great flexibility: you can log information (and errors) to a file, a database, a message queue, the event log, or just generate an email. If you're trying to choose between the two, consider these points:

- Enterprise Library has a GUI tool for configuring its Logging Application Block
- log4net supports hierarchical log maintenance

The choice is based mainly on features you need to address because, from a performance point of view, they're similar. Enterprise Library is often used because of its capabilities. If you're using it already in your project (for example, because you're using the Cache Application Block), it might seem familiar to you; in this case, using the Enterprise Library is the right choice because you already have a dependency on the main library. On the other hand, log4net is preferred by developers who are searching for a simple and complete library to perform this task, and nothing more.

If you prefer to write code, and your logging needs are relative only to exceptions, you'll probably find it easier to handle errors and store this information with your own custom code.

TECHNIQUE 92 Intercepting, and handling errors with a custom module

Exposing errors to end users isn't a good idea, from both a usability and a security point of view. Error handling implemented the right way will help administrators to inspect the complete error, and will provide the casual user with a more useful courtesy page.

PROBLEM

You want to avoid full error disclosure to normal users but display the full error to administrators. Your application will be secure, and administrators will be able to inspect errors, without accessing the error logging tool, while they're running the page that caused the error. You also want to provide an entry point to add more powerful exception logging capabilities in the future.

SOLUTION

As we discussed in chapter 4 when we talked about security, it's important not to show sensitive information to users: you should always consider errors to be dangerous. ASP.NET gives you control over errors, letting you choose from three options:

- Always show errors
- Never show errors
- Show errors only when the request is coming from the same machine that's running the application

Following code comes from a typical web.config and demonstrates each of these options:

```
<configuration>
  <system.web>
    <customErrors mode="On|Off|RemoteOnly"
                  defaultRedirect="CustomPage.htm" />
  </system.web>
</configuration>
```

These settings are flexible enough to cover your needs while you're developing the application. The reality is that when you put your application in production, you probably won't make requests from the same machine running the page, and so you need to be the only one accessing error details.

`HttpApplication` has a useful `Error` event, used to intercept exceptions that aren't blocked at a higher level, such as in a try/catch block. This event can be handled to combine authorization and authentication from ASP.NET—you can show the error to only a specific group of people, thanks to the Roles API that's available on ASP.NET (see chapter 5 for more information about the Roles API). The code is simple: you just have to handle the event, verify user permissions given the current roles, and then show a personalized error page—or just let ASP.NET do the magic, using the values specified in web.config.

We need to configure web.config to register our module, just like we did in listing 15.1. When an error occurs, the exception will be handled by our event handler, and we'll display an error message similar to the one shown in figure 15.3.

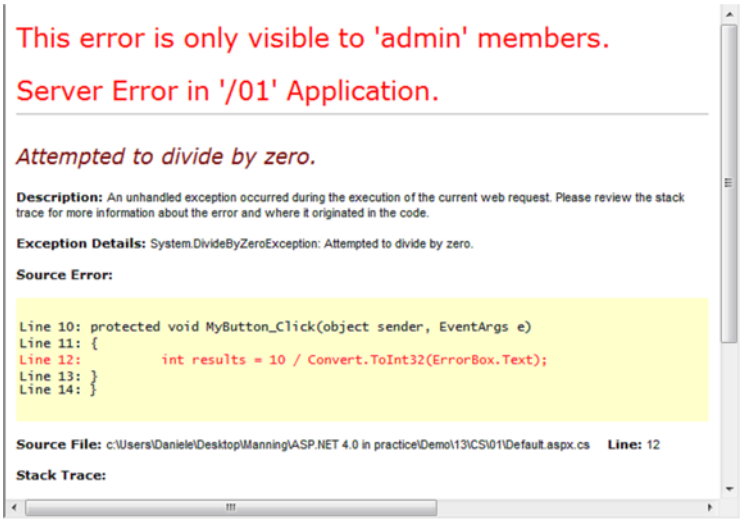

Figure 15.3
Using our custom error system, we can add additional information to the error page or decide to show the error to specific clients.

To implement such a personalized view, we need to write a custom `HttpModule` like the one shown in the following listing.

Listing 15.3 A custom error logging module

C#:

```
namespace ASPNET4InPractice.
{
  public class ErrorModule: IHttpModule
  {
    ...
    public void Init(HttpApplication context)
    {
      context.Error+=new EventHandler(OnError);
    }

    void OnError(object sender, EventArgs e)
    {
      HttpApplication app = (HttpApplication)sender;
      HttpException ex = app.Server.GetLastError() as HttpException;

      if (app.User.IsInRole(AdministrativeRole))
      {
        app.Response.Clear();
        app.Response.TrySkipIisCustomErrors = true;
        app.Response.Write(
             string.Format("<h1>This error is only visible" +
                           " to '{0}' members.</h1>", AdministrativeRole));
        app.Response.Write(ex.GetHtmlErrorMessage());
        app.Context.ApplicationInstance.CompleteRequest();      }
    }
  }
}
```

Register for Error event on HttpApplication

Display error details

VB:

```
Namespace ASPNET4InPractice
  Public Class ErrorModule
    Implements IHttpModule
    ...
    Public Sub Init(ByVal context As HttpApplication)
➥         Implements IHttpModule.Init
      AddHandler context.Error, AddressOf OnError
    End Sub

    Private Sub OnError(ByVal sender As Object, ByVal e As EventArgs)
      Dim app As HttpApplication = DirectCast(sender, HttpApplication)
      Dim ex As HttpException = TryCast(app.Server.GetLastError(),
                                        HttpException)

      If app.User.IsInRole(AdministrativeRole) Then
        app.Response.Clear()
        app.Response.TrySkipIisCustomErrors = True

        app.Response.Write(String.Format("<h1>This error is only visible" &
                           " to '{0}' members.</h1>", AdministrativeRole))
        app.Response.Write(ex.GetHtmlErrorMessage())
        app.Context.ApplicationInstance.CompleteRequest()      End If
```

Register for Error event on HttpApplication

Display error details

```
    End Sub
  End Class
End Namespace
```

You can easily adapt this code to integrate more logging instrumentations, like form variables or application status. To register the module, you have to place this configuration in your web.config:

```
<configuration>
  <appSettings>
    <add key="admnistrativeRole" value="admin"/>
  </appSettings>
  <system.web>
    <httpModules>
      <add name="CustomErrorModule"
           type="ASPNET4InPractice.Chapter15.ErrorModule, App_Code"/>
    </httpModules>

    <customErrors mode="On" defaultRedirect="ErrorPage.htm" />
  </system.web>
</configuration>
```

Sending error message details via email

If you want to send every error via email, the `Error` event handler is the right place to add your code. You can use the `MailMessage` class from `System.Net.Mail` to compose a notification email and send it to your address. If you want to use something already available, take a look at Health Monitoring in the MSDN documentation at http://www.mng.bz/8p51. If you want to store the error log in a database table or in a file, see the corresponding topics in chapters 2 and 3.

Use the `TrySkipIisCustomErrors` property from the `HttpResponse` class to modify the default behavior of IIS 7.x when you're dealing with custom errors. By default, IIS 7 bypasses local error handling and, instead, uses the configuration applied in the `system.webServer` section. By setting this property, you can control IIS 7.x behavior, too; the behavior of IIS 6.0 isn't affected by this change.

DISCUSSION

`HttpModules` enable global event handling and are useful whenever you have that kind of situation. This approach is simple, centralized, open to additional improvements, and shows you how easy it is to tweak ASP.NET behavior and avoid security concerns at the same time. You can handle error logging with many different approaches, as well as with the libraries we mentioned earlier. The methods we've described here are a starting point. The main thing to keep in mind no matter how you decide to deal with the problem is that the less an attacker sees, the better your application security is.

Our journey through ASP.NET advanced techniques will continue now with a topic that's bound to be of interest to you: how to extend ASP.NET `HttpRuntime` and gain more control over ASP.NET page compilation.

15.3 Extending ASP.NET HttpRuntime

ASP.NET `HttpRuntime` provides great flexibility. If you need to tweak something related to ASP.NET, you'll probably end up with `HttpRuntime`. Both `HttpHandlers` and `HttpModules` are considered part of `HttpRuntime`, but you can leverage other things to modify ASP.NET.

`VirtualPathProvider` is a feature that was introduced with ASP.NET 2.0. You can use it to dynamically load resources from a source that's different from that of the file-system and to build them as if they were normal resources. `VirtualPathProvider` is intended for browsable resources (.aspx, .ascx, master pages, and themes). If you want to virtualize other kinds of resources, you need to implement a `BuildProvider`.

`VirtualPathProvider` must be registered at application startup, usually in the `AppInitialize` static method with global.asax or in the constructor of an `HttpModule`. Unfortunately, `VirtualPathProvider` won't work with a precompiled web site, unless you try some of the hacks that use reflection to invoke a private method. That scenario isn't tested, so try it at your own risk.

TECHNIQUE 93 Running your site from the database

Running your code from the database is easy using `VirtualPathProvider`. You can define special kinds of requests to be served from a database, so maintenance will be simpler and won't require FTP access. Microsoft Office SharePoint Server (MOSS) is built on that assumption, so you'll probably find this technique useful in your projects, too.

PROBLEM

Saving the page source on disk is feasible for many of the situations you'll face. But in some cases, you might need to store it at other locations, such as a database, without any loss of features. This solution might be useful when you have multiple servers and you need to keep the source in sync among different servers, without using network shares or something similar.

SOLUTION

`VirtualPathProvider` is built on top of three fundamental classes that come from the `System.Web.Hosting` namespace:

- `VirtualPathProvider`—Used as a base class for the implementation
- `VirtualDirectory`—Represents a directory
- `VirtualFile`—Represents a file

> **NOTE** Custom implementations for the `VirtualPathProvider`, `VirtualDirectory`, and `VirtualFile` classes need to run under full trust permissions. If you're using another trust level, you can't run this example. For more information on trust levels, see http://mng.bz/cuH6.

First of all, we need to implement a new class that derives from `VirtualPathProvider` and overrides the `FileExists` and `DirectoryExists` methods. These methods are

used to determine whether the requested file or directory exists. The `GetFile` and `GetDirectory` methods are implemented to serve an instance of `VirtualFile` and `VirtualDirectory`, respectively. These classes represent the files and directories, and you use them even with normal files and directories coming from the filesystem. You'll get the same experience, but your code will be loaded from the database. The difference in this scenario is that we need to implement these classes to represent our concepts of both directories and files.

Our custom `VirtualDirectory` implementation isn't difficult: we simply need to implement a class similar to the one shown in the following listing.

Listing 15.4 Our VirtualDirectory implementation

C#:

```
namespace ASPNET4InPractice
{
  public class DatabaseDirectory : VirtualDirectory
  {
    private List<string> _directories = new List<string>();
    private List<string> _files = new List<string>();
    private List<string> _children = new List<string>();

    public DatabaseDirectory(string virtualPath): base(virtualPath) {}

    public override IEnumerable Children
    {
      get
      {
        return _children;
      }
    }

    public override IEnumerable Directories
    {
      get
      {
        return _directories;
      }
    }

    public override IEnumerable Files
    {
      get
      {
        return _files;
      }
    }
  }
}
```

Directories in path

Files in path

VB:

```
Namespace ASPNET4InPractice
  Public Class DatabaseDirectory
    Inherits VirtualDirectory
    Private _directories As New List(Of String)()
```

```
    Private _files As New List(Of String)()
    Private _children As New List(Of String)()

    Public Sub New(ByVal virtualPath As String)
      MyBase.New(virtualPath)
    End Sub

    Public Overloads Overrides ReadOnly Property Children() As IEnumerable
      Get
        Return _children
      End Get
    End Property

    Public Overloads Overrides ReadOnly Property Directories()
➥       As IEnumerable                                          <-- Directories in path
      Get
        Return _directories
      End Get
    End Property

    Public Overloads Overrides ReadOnly Property Files()
➥       As IEnumerable                                          <-- Files in path
      Get
        Return _files
      End Get
    End Property
  End Class
End Namespace
```

A `VirtualFile` implementation is more difficult to pull off because we need to get the file content from the database. We'll use a table like the one in figure 15.4 to represent our virtual filesystem. We'll use Entity Framework to map our table to an object model and query it using LINQ extensions methods.

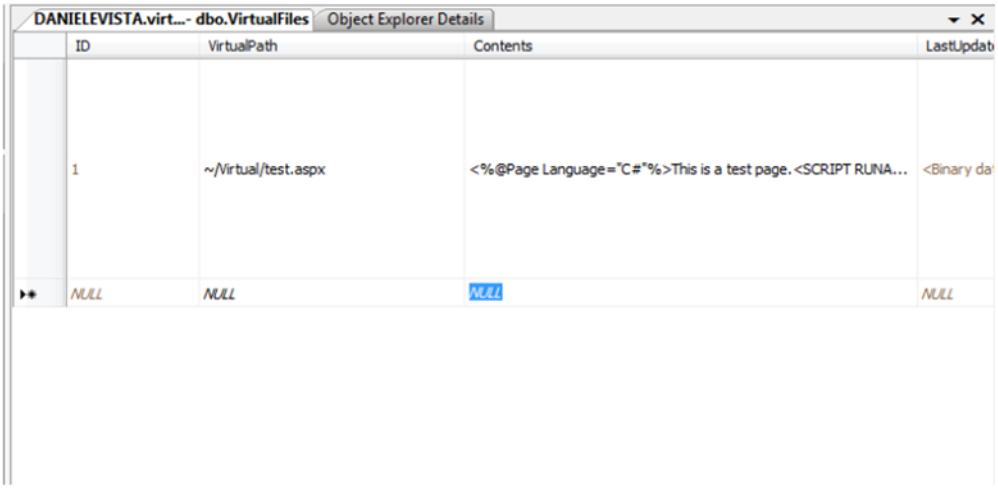

Figure 15.4 The database model used to represent our virtual filesystem is simple and consists of three columns to identify the page path, its content, and the last modified date.

The core of this system is shown in the following listing, which contains a snippet from the `VirtualFile` implementation details.

Listing 15.5 VirtualFile implementation to load content from a database

C#:

```
namespace ASPNET4InPractice
{
  public class DatabaseFile : VirtualFile
  {
    public DatabaseFile(string virtualPath) : base(virtualPath) { }

    public override Stream Open()
    {
      // get file contents and write to the stream
      string fileContents = Utility.GetFileContents(
             VirtualPathUtility.ToAppRelative(VirtualPath));

      Stream stream = new MemoryStream();
      if (!string.IsNullOrEmpty(fileContents))
      {
        StreamWriter writer = new StreamWriter(stream);
        writer.Write(fileContents);
        writer.Flush();
        stream.Seek(0, SeekOrigin.Begin);
      }
      return stream;
    }
  }
}
```

Retrieve file contents

Read content in memory

VB:

```
Namespace ASPNET4InPractice
    Public Class DatabaseFile
        Inherits VirtualFile
        Public Sub New(ByVal virtualPath As String)
            MyBase.New(virtualPath)
        End Sub

        Public Overloads Overrides Function Open() As Stream
            ' get file contents and write to the stream
            Dim fileContents As String = Utility.GetFileContents(
                VirtualPathUtility.ToAppRelative(VirtualPath))

            Dim stream As Stream = New MemoryStream()
            If Not String.IsNullOrEmpty(fileContents) Then
                Dim writer As New StreamWriter(stream)
                writer.Write(fileContents)
                writer.Flush()
                stream.Seek(0, SeekOrigin.Begin)
            End If
            Return stream
        End Function
    End Class
End Namespace
```

Retrieve file contents

Read content in memory

To check whether a file is modified, we'll add a new method that contains this code:

C#:

```
public byte[] LastModifiedTimeStamp
{
  get
  {
    return Utility.GetLastModifiedTimeStamp(
                VirtualPathUtility.ToAppRelative(VirtualPath));
  }
}
```

VB:

```
Public ReadOnly Property LastModifiedTimeStamp() As Byte()
  Get
    Return Utility.GetLastModifiedTimeStamp(
           VirtualPathUtility.ToAppRelative(VirtualPath))
  End Get
End Property
```

The last thing to implement is the real `VirtualPathProvider` custom class. We need to derive from this class and override a couple of methods:

- `GetCacheDependency` and `GetFileHash`—Implemented to provide a custom mechanism for cache dependency. ASP.NET uses a simple method to determine whether a given resource needs to be recompiled or the current one can be used. Our custom implementation has no `CacheDependency` but does provide a custom-computed `HashCode`, using a timestamp column in the database
- `FileExists` and `DirectoryExists`—These methods are used to determine whether a file or directory exists. For directories, we simply return `true` if the path is inside our scope. For files, we check the existence of the virtual path using our Entity Framework model.
- `GetFile` and `GetDirectory`—Get the corresponding `VirtualFile` and `VirtualDirectory` custom implementations, which are included in the downloadable code.

The code in the following listing contains the main methods used to retrieve the file and directory content.

Listing 15.6 The VirtualPathProvider implementation

C#:

```
namespace ASPNET4InPractice
{
  public class DatabasePathProvider : VirtualPathProvider
  {
    public DatabasePathProvider() : base()
    {

    public override VirtualFile GetFile(string virtualPath)
    {
      if (IsVirtualPath(virtualPath))
        return new DatabaseFile(virtualPath);
```

Load content from database

```
      else
        return Previous.GetFile(virtualPath);
    }

    public override VirtualDirectory GetDirectory(string virtualDir)
    {
      if (IsVirtualPath(virtualDir))
        return new DatabaseDirectory(virtualDir);
      else
        return Previous.GetDirectory(virtualDir);
    }

  }
}
```

Use previous provider

Load content from database

Use previous provider

VB:

```
Namespace ASPNET4InPractice
  Public Class DatabasePathProvider
    Inherits VirtualPathProvider
    Public Sub New()
      MyBase.New()
    End Sub

    Public Overloads Overrides Function GetFile(
         ByVal virtualPath As String) As VirtualFile
      If IsVirtualPath(virtualPath) Then
        Return New DatabaseFile(virtualPath)
      Else
        Return Previous.GetFile(virtualPath)
      End If
    End Function

    Public Overloads Overrides Function GetDirectory(
         ByVal virtualDir As String) As VirtualDirectory
      If IsVirtualPath(virtualDir) Then
        Return New DatabaseDirectory(virtualDir)
        Return Previous.GetDirectory(virtualDir)
      End If
    End Function
  End Class
End Namespace
```

Load content from database

Use previous provider

Load content from database

Use previous provider

To verify that a file or directory exists, you have to implement the `FileExists` and `DirectoryExists` methods. The `VirtualPathUtility.ToAppRelative` method is used to convert the absolute path to an application-relative one (for example, /myroot/Virtual/test2.aspx to ~/Virtual/test2.aspx), as in the following listing.

Listing 15.7 FileExists and DirectoryExists implementation

C#:

```
public override bool FileExists(string virtualPath)
{
  if (IsVirtualPath(virtualPath) &&
        Utility.FileExists(
                      VirtualPathUtility.ToAppRelative(virtualPath)))
    return true;
```

```
  return Previous.FileExists(virtualPath);
}

public override bool DirectoryExists(string virtualDir)
{
  if (IsVirtualPath(virtualDir))
    return true;

  return Previous.DirectoryExists(virtualDir);
}
```

VB:

```
 Private Function IsVirtualPath(ByVal virtualPath As String) As Boolean
      Return VirtualPathUtility.ToAppRelative(virtualPath).
                              StartsWith(Utility.BasePath,
                                StringComparison.InvariantCultureIgnoreCase)
    End Function

Public Overloads Overrides Function FileExists(
                ByVal virtualPath As String) As Boolean
      If IsVirtualPath(virtualPath) AndAlso
          Utility.FileExists(VirtualPathUtility.ToAppRelative(virtualPath))
      Then
        Return True
      End If

      Return Previous.FileExists(virtualPath)
    End Function

    Public Overloads Overrides Function DirectoryExists(
                ByVal virtualDir As String) As Boolean
      If IsVirtualPath(virtualDir) Then
        Return True
      End If

      Return Previous.DirectoryExists(virtualDir)
    End Function
```

Last, but not least, to avoid performance loss, you must cache the compilation. `VirtualPathProvider` has two methods that control caching behavior, as implemented in the following listing.

Listing 15.8 Controlling page compilation in VirtualPathProvider

C#:

```
public override CacheDependency GetCacheDependency(string virtualPath,
                IEnumerable virtualPathDependencies, DateTime utcStart)
{
  if (IsVirtualPath(virtualPath))
    return null;

  return base.GetCacheDependency(virtualPath,
                    virtualPathDependencies, utcStart);
}

public override string GetFileHash(string virtualPath,
         IEnumerable virtualPathDependencies)
```

Implementation default

Invalidate compilation

```
{
  HashCodeCombiner hashCodeCombiner = new HashCodeCombiner();

  List<string> unrecognizedDependencies = new List<string>();

  foreach (string virtualDependency in virtualPathDependencies)
  {
    if (IsVirtualPath(virtualDependency))
    {
      DatabaseFile file = (DatabaseFile)GetFile(virtualDependency);
      hashCodeCombiner.AddObject(file.LastModifiedTimeStamp);
    }
    else
    {
      unrecognizedDependencies.Add(virtualDependency);
    }
  }

  string result = hashCodeCombiner.CombinedHashString;

  if (unrecognizedDependencies.Count > 0)
  {
    result += Previous.GetFileHash(virtualPath, unrecognizedDependencies);
  }

  return result;
}
```

VB:

```
Public Overloads Overrides Function GetCacheDependency(
                                      ByVal virtualPath As String,
                    ByVal virtualPathDependencies As IEnumerable,
                    ByVal utcStart As DateTime) As CacheDependency
  If IsVirtualPath(virtualPath) Then
    Return Nothing
  End If

  Return MyBase.GetCacheDependency(
                     virtualPath,
                     virtualPathDependencies,
                     utcStart)
End Function

Public Overloads Overrides Function GetFileHash(
                                            ByVal virtualPath As String,
                ByVal virtualPathDependencies As IEnumerable) As String
  Dim hashCodeCombiner As New HashCodeCombiner()

  Dim unrecognizedDependencies As New List(Of String)()

  For Each virtualDependency As String In virtualPathDependencies
    If IsVirtualPath(virtualDependency) Then
      Dim file As DatabaseFile = DirectCast(
                       GetFile(virtualDependency), DatabaseFile)
      hashCodeCombiner.AddObject(file.LastModifiedTimeStamp)
    Else
      unrecognizedDependencies.Add(virtualDependency)
    End If
  Next

  Dim result As String = hashCodeCombiner.CombinedHashString
```

Implementation default

Invalidate compilation

```
  If unrecognizedDependencies.Count > 0 Then
    result += Previous.GetFileHash(virtualPath, unrecognizedDependencies)
  End If

  Return result
End Function
```

The code used in this solution isn't hard to understand, but it is a bit verbose; `VirtualPathProvider` and its relative classes are general purpose and need a lot of plumbing code to be implemented. As we've shown you in this example, you don't need to fully implement all of them, but only the one that's useful in your scenario.

DISCUSSION

The scenario addressed in this section provided a good lesson in how extensible ASP.NET is. The code isn't difficult to understand and it uses some of the inner details of `HttpRuntime`. You can apply this code in multiple ways, beginning with using a different store for the file source to enable better administration and to simplify code distribution across load-balanced web servers. As you can see in figure 15.5, the result will look as if the page was stored on a disk and then traditionally executed.

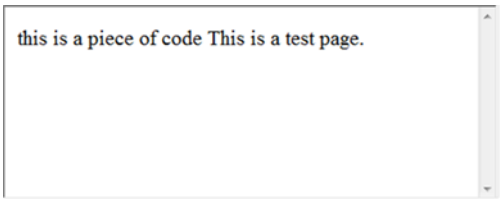

Figure 15.5 The resulting page is served from the database that was previously created. When the database content is changed, the new version is automatically used.

If you plan to use this code in production, you'll probably need to add some caching to improve performance and save your database from a few calls. We've attached a simple page (admin.aspx) to let you experiment easily by administering the virtual file-system content.

This example is the last one in the chapter because it deals with compilation, the Page Parser, and the inner details of ASP.NET engine. We hope it's useful to you and has shown you how easily you can enhance ASP.NET to suit your needs.

15.4 Summary

This chapter contained special techniques to address specific scenarios. Remember that ASP.NET is built for flexibility; this characteristic reflects how many incredible things you can do by using extreme techniques.

`HttpRuntime` offers the right entry points to add your own custom mechanisms to implement simple things like intercepting (and logging) exceptions and more complex things like modifying page behavior using an `HttpModule`, or implementing a `VirtualPathProvider` to load code from a database. ASP.NET is so powerful that you can literally do anything you need to: you just have to write code and express your imagination!

The next chapter, which is the last one in the book, will continue on this path and introduce you to useful tips that will increase the performance of your ASP.NET applications.

.NET DEVELOPMENT

ASP.NET 4.0 IN PRACTICE

Bochicchio • Mostarda • De Sanctis

ASP.NET is a massive framework that requires a large amount of know-how from developers. Fortunately, this book distills over 100 practical ASP.NET techniques from the experience of a team of MVPs, and puts them right at your fingertips.

The techniques are tested and selected for their usefulness, and they are all presented in a simple problem-solution-discussion format. You'll discover methods for key new subjects like data integration with Entity Framework and ASP.NET MVC. Along the way, you'll also find ways to make your applications fast and secure.

What's Inside

- The Identity Map pattern in EF 4
- Use Master Pages to define a common UI
- Adaptive Rendering
- Save user login data securely
- ... and much more

This book is written for developers familiar with the basics of ASP .NET, looking to become more productive with it.

Daniele Bochicchio, Stefano Mostarda, and Marco De Sanctis are ASP .NET MVPs and core members of ASPItalia.com, Italy 's largest .NET community. They are also the authors of Manning's *Entity Framework 4 in Action*.

For access to the book's forum and a free ebook for owners of this book, go to www.manning.com/ASP.NET4.0inPractice

"The right book to sharpen your ASP.NET skills."
—Alessandro Gallo
Microsoft MVP

"Easy to read, full of extremely helpful techniques."
—David Barkol, Neudesic

"A great way to learn an exciting new technology."
—Gary A. Bushey
ShareTech Consulting

"Get up to speed in no time."
—Nikander and Margriet Bruggeman
Lois & Clark IT Services

ISBN-13: 978-1-935182-46-7
ISBN-10: 1-935182-46-3
55499
9 781935 182467

MANNING $54.99 / Can $63.99 [INCLUDING eBOOK]