10 Konfiguration, Optimierung und Sicherheit

Die Themen Konfiguration, Optimierung und Sicherheit spielen bei der täglichen Programmierung keine direkte Rolle, denn es handelt sich meist um einmalige Aufgaben. Das ändert nichts an der Bedeutung, die sie für stabile Applikationen haben. Dieses Kapitel gibt Anregungen, welche Methoden es in ASP.NET gibt und welche elementaren Schritte erforderlich sind.

10.1 Schnellstart

Dieser Abschnitt gibt einen kompakten Überblick über das Thema und zeigt sinnvolle Verknüpfungen mit ergänzenden und vorbereitenden Kapiteln. Der Wegweiser in die Referenz hilft, die passenden Seiten in der MSDN-Online-Referenz besonders schnell zu finden.

10.1.1 Über dieses Kapitel

In diesem Kapitel erfahren Sie mehr über die Konfiguration von Seiten, Applikationen und Servern. ASP.NET basiert auf einem dreistufigen Konfigurationsmodell. Durch so genannte Direktiven kann das Verhalten einer *aspx*-Seite gesteuert werden. Für Benutzer-Steuerelemente und Web-Services gibt es spezielle Direktiven. Abschnitt 10.2, »Konfiguration einzelner Seiten: Die Direktiven« ab Seite 954 behandelt dieses Thema.

Die Konfiguration einer Applikation basiert – ebenso wie die der gesamten Maschine – auf Konfigurationsdateien. Diese liegen im XML-Format vor und lassen sich sehr leicht bearbeiten. Einen Überblick finden Sie in Abschnitt 10.3, »Konfiguration von Appli-

kationen: web.config« ab Seite 961. Weitere Details sind jedoch in den Abschnitten des Buches zu finden, in denen bestimmte Teilfunktionen konfiguriert werden müssen.

Die Sicherheit von Webanwendungen und Servern spielt heute eine herausragende Rolle. ASP.NET und das Framework sind hier gut gewappnet. Abschnitt 10.5, »Sicherheit« ab Seite 972 führt in die Techniken der Absicherung von Seiten und Applikationen ein. Sie finden dort auch Informationen über die Programmierung von Anmeldeformularen, die Nutzung der integrierten Sicherheit von Windows 2000/.NET-Server und Methoden der Wiedererkennung von Benutzern.

10.2 Konfiguration einzelner Seiten: Die Direktiven

Viele grundlegende Operationen lassen sich über Seitendirektiven steuern. Diese stehen am Anfang der Seite und gelten für Übersetzung oder Ausführung. Die Direktiven sollten immer an den Anfang der Seite gesetzt werden, auch wenn dies nicht in allen Fällen zwingend notwendig ist.

10.2.1 Übersicht

Folgende Direktiven sind verfügbar:

- `@Page`

  Hiermit werden Vorgänge gesteuert, die unmittelbar mit der Ausführung der Seite zu tun haben.

- `@Control`

  Mit dieser Direktive werden Benutzer-Steuerelemente kontrolliert. Die Attribute sind mit `@Page` vergleichbar.

- `@Import`

  Mit dieser Direktive werden Namensräume aus dem .NET-Framework importiert. Dies ist notwendig, wenn Code Behind nicht verwendet wird, da in *aspx*-Seiten die Anweisung `using` nicht eingesetzt werden kann, mit der üblicherweise Namensräume importiert werden.

- `@Implements`

  Diese Direktive importiert eine Schnittstellenbeschreibung (Interface) aus dem .NET-Framework.

- `@Register`

  Hiermit werden Benutzer-Steuerelemente (User Controls) angemeldet, sodass der Compiler diese finden und einbinden kann.

- `@Assembly`

  Die Nachfolger der DLLs in Windows sind in .NET Assemblies. Sie enthalten praktisch bereits übersetzten Code in der Intermediate Language. Mit dieser Direktive können Sie Assemblies direkt nutzen.

- `@OutputCache`

  Jede Seite, die auf dem Server erzeugt wird, kann zwischengespeichert werden. Dies erhöht die Systemleistung und verringert die Prozessorlast. Für den korrekten Aufbau der Seite im Browser ist die die Kontrolle dieses Vorgangs wichtig. Diese Direktive steuert dies.

- `@Reference`

  Dieses Element stellt für die Verbindung von Seiten bei der Steuerungsweitergabe mit `Server.Transfer` die Referenz her.

Die wichtigsten Direktiven werden nachfolgend vorgestellt – in dem Rahmen, indem sie in diesem Buch Verwendung finden. In einigen Fällen erfolgte eine genauere Erläuterung im Zusammenhang mit der Anwendung. Dann wird auf die entsprechenden Abschnitte verwiesen.

10.2.2 Die Seitendirektive @Page

Die Seitendirektive `@Page` definiert Bedingungen zur Abarbeitung einer Seite. Sie steht immer am Anfang einer *aspx*-Seite:

```
<% @Page Attribut="Wert" ... %>
```

Die wichtigsten Attribute finden Sie in der folgenden Tabelle:

Attribut	Parameter	Bedeutung
AutoEventWireUp	true, false	Leitet Ereignisse automatisch weiter, siehe Text nach der Tabelle.
Buffer	true, false	Steuert den Ausgabepuffer ein. true ist der Standardwert.
ClassName	Name	Name einer Klasse, die für diese Seite übersetzt wird, wenn mehrere Klassen zu Auswahl stehen.
ContentType	z. B. image/gif	Der Name eines MIME-Typs, mit dem der Inhalt gesendet werden soll.
Debug	true, false	Schaltet serverseitiges Debuggen ein. Der Standardwert ist false (aus).
Trace	true, false	Schaltet serverseitiges Ablaufverfolgung ein. Der Standardwert ist false (aus).
Description	Text	Infotext zur Seite, der von ASP.NET ignoriert wird.
Inherits	Name	Name einer Klasse einer hinterlegten Code-Datei (Code Behind).
Src	Name, URL	Quelle einer Code-Datei (Code Behind)
Language	C#, VB, JScript.NET	Name der Sprache, die im Inline-Code (<%%>-Tags) erwartet wird.
Culture	Name	Name des Sprachpaketes, z. B. »de-DE«, zur Einstellung der Währung, Zahlenformate usw.

Tabelle 10.1: Die wichtigsten Attribute der Direktive `@Page`

AutoEventWireUp

Die Aktivierung der automatischen Ereignisverarbeitung ist notwendig, wenn Sie mit der im Buch überwiegend verwendeten Technik arbeiten, bei der Ereignisbehandlungsmethoden lediglich als `public` deklariert und dann automatisch vom Ereignis eines Steuerelements gefunden werden. Der Visual Studio .NET-Designer arbeitet anders. Er hängt die Ereignisse in einer gesonderten Initialisierungsmethode selbst an die Delegaten an. Damit können diese als `private` gekennzeichnet werden.

Einsatz der Seitendirektive zur Fehlersuche

Debug

Die Attribute `Debug` und `Trace` helfen bei der Fehlersuche. Mit Debug wird der Ablauf des Programms auf der Serverseite überwacht. Tritt ein Fehler auf, kann die ASP.NET detaillierte Informa-

tionen über die Ursache bereitstellen. In Produktionsumgebungen wird das Attribut entfernt, weil die Überwachung Leistung kostet. Wenn Sie in manchen Situationen unzureichende Informationen über den Fehler finden oder die Angabe der Zeile mit der Ursache fehlt, fügen Sie der Seitendirektive `debug="true"` hinzu.

Wenn Sie noch mehr Informationen über den Gesamtzustand des Systems benötigen, um die Ursache eines Laufzeitfehlers zu finden, ergänzen Sie das Attribut `trace="true"`.

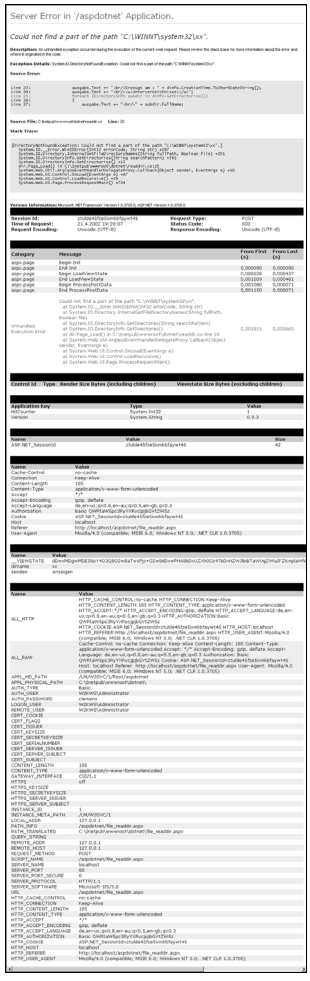


Abbildung 10.1: Ausgabe eines Laufzeitfehlers mit `trace="true"`

10.2.3 Die Direktive @Import

Mit dieser Direktive werden weitere Namensräume eingebunden. Sie kann so oft angewendet werden, wie Namensräume benötigt werden:

```
<% @Import namespace="System.Data" %>
```

ASP.NET importiert einige Namensräume automatisch, sodass die Anwendung nicht immer notwendig ist. Dies gilt nur für die Seite selbst, nicht für hinterlegten Code.

Folgende Namensräume werden eingebunden:

- System
- System.Collections
- System.Collections.Specialized
- System.Configuration
- System.IO
- System.Text
- System.Text.RegularExpressions
- System.Web
- System.Web.Caching
- System.Web.Security
- System.Web.SessionState
- System.Web.UI
- System.Web.UI.HtmlControls
- System.Web.UI.WebControls

Wenn `@Import` erforderlich ist, wird es in diesem Buch explizit erwähnt.

10.2.4 Die Direktive @Register

Mit dieser Direktive werden Benutzer-Steuerelemente (User Controls) der Seite bekannt gemacht:

```
<% @Register tagprefix="tagprefix" Tagname="tagname" Src="pathname" %>
```

Festgelegt wird der Präfix, den das Element auf der Seite verwendet, der Name und der Pfad, wo die *ascx*-Datei gespeichert wurde.

Eine Anwendung finden Sie unter anderem im Abschnitt 7.2, »Modularer Code mit Benutzer-Steuerelementen« ab Seite 654.

Attribute und Parameter	Beschreibung
`TagPrefix`	Präfix des Steuerelements, der als Alias für den Namensraum gilt
`TagName`	Name des Steuerelements
`NameSpace`	Namensraum, aus dem das Element stammt. Die Angabe ist optional.
`Assembly`	Name einer Assembly, die die Definition enthält (Angabe ohne den Suffix ».dll«). Dieses Attribut kann nicht gleichzeitig mit `Path` verwendet werden.
`Path`	Pfad, unter dem das Element definiert wurde (Angabe vollständig mit dem Suffix »ascx«). Dieses Attribut kann nicht gleichzeitig mit `Assembly` verwendet werden.

10.2.5 Die Direktive @OutputCache

Die Direktive `@OutputCache` steuert das Verhalten des Ausgabezwischenspeichers. Der Zugriff auf ganz oder teilweise statische Inhalte kann damit beschleunigt werden. Eine nähere Erläuterung der Verfahrensweise finden Sie in Abschnitt 8.7.1, »Caching von Seiten und Steuerelementen« ab Seite 814.

Die folgende Tabelle enthält die zulässigen Attribute und deren Bedeutung.

Attribute und Parameter	Beschreibung
`Duration`	Dauer, während der die Seite oder das Steuerelement im Cache verbleibt. Angabe in Sekunden.
`Location`	Optional, nicht für Steuerelemente verwendbar. Bestimmt, welcher physische Speicherort verwendet wird: ▶ `Any` Jeder verfügbare Speicher wird verwendet. ▶ `Client` Der Speicher des Browsers wird verwendet

Tabelle 10.2: Die Attribute der Direktive `@OutputCache`

Attribute und Parameter	Beschreibung
	▶ `DownStream` Der Speicher des Browsers oder ein HTTP 1.1 fähiger Proxy wird verwendet ▶ `None` Die Zwischenspeicherung ist deaktiviert ▶ `Server` Der Ausgabespeicher befindet sich auf dem Webserver
`VaryByCustom`	Optional. Entweder die Zeichenkette »browser«, dann varriert der Cache nach dem Client, d. h., jeder Browser oder eine beliebige Zeichenkette, dann ist eine Änderung der Standard-Cachemethoden erforderlich.
`VaryByHeader`	Optional. Liste von HTTP-Kopfzeilen, die benutzerdefinierte Anforderungen an das Speichergerät enthalten
`VaryByParam`	Erforderlich. Liste von Zeichenfolgen, die POST- oder GET-Parameter beschreiben, die die Änderung des Cache steuern. Sonderfälle sind: ▶ `none` keine Berücksichtigung von Parametern ▶ `*` Alle gültigen Parameter werden unterschieden
`VaryByControl`	Name des Steuerelements, auf das sich der Cache bezieht. Nur zulässig im Steuerelemente-Caching.

Tabelle 10.2: Die Attribute der Direktive `@OutputCache` (Forts.)

Wenn Listen in den Parametern verwendet werden dürfen, sind die Listenelemente durch Kommata zu trennen, soweit nichts anderes in der Beschreibung genannt wird.

10.2.6 Die Direktive @Control

Wenn Sie Benutzer-Steuerelemente entwerfen, erhalten auch diese eine mit `@Page` vergleichbare Direktive: `@Control`. Die folgende Tabelle zeigt alle zulässigen Attribute und deren Bedeutung:

Attribute und Parameter	Beschreibung
`AutoEventWireUp="true|false"`	Aktiviert die automatische Ereignisverarbeitung
`ClassName="Name"`	Name der Klasse, in die das Steuerelement kompiliert werden soll
`CompilerOptions=" "`	Compileroptionen
`Debug="true|false"`	Schaltet Debugging ein
`Description="Beschreibung"`	Beschreibung des Steuerelements
`EnableViewState="true|false"`	Schaltet den Anzeigestatus ein oder aus.
`Inherits="KlassenName"`	Name der zu verwendenden Klasse aus der hinterlegten Code-Datei
`Src="CodeDatei"`	Name der Code-Datei
`Language="c#"`	Sprache des eingebetteten Codes
`WarningLevel="0|1|2|4"`	Reaktionsstufen des Compilers

Tabelle 10.3: Die Attribute der Direktive `@Control`

AutoEventWireUp

Die Aktivierung der automatischen Ereignisverarbeitung ist notwendig, wenn Sie mit der im Buch überwiegend verwendeten Technik arbeiten, bei der Ereignisbehandlungsmethode lediglich als `public` deklariert werden und dann automatisch vom Ereignis eines Steuerelements gefunden werden. Der Visual Studio .NET-Designer arbeitet anders. Er hängt die Ereignisse in einer gesonderten Initialisierungsmethode selbst an die Delegaten an. Damit können diese als `private` gekennzeichnet werden.

Die nur für VB.NET zutreffenden Optionen wurden hier nicht aufgeführt.

10.3 Konfiguration von Applikationen: web.config

Um eine Applikation zu konfigurieren, wird mit ASP.NET ein ebenso einfaches wie leistungsfähiges Prinzip eingeführt. Neben einer zentralen Konfigurationsdatei, *machine.config*, die im nächsten Abschnitt beschrieben wird, kann der Betreiber einer Applikation in jedem Verzeichnis eine Datei *web.config* abgelegen. Diese XML-Datei enthält eine einfache Struktur von Tags, mit denen vielfältige Konfigurationen möglich sind. Um die Aktivierung müssen Sie sich nicht kümmern. Es wird weder ein Neustart der Applikation

noch des IIS gefordert. Tatsächlich lädt ASP.NET die Datei beim ersten Aufruf und stellt den Inhalt dann aus dem Speicher zur Verfügung. Nur wenn sich Änderungen ergeben, wird sie neu geladen. Insofern ist die ständige Nutzung nicht mit Festplattenzugriffen verbunden, Änderungen wirken sich aber dennoch sofort aus.

Das vollständige Schema der Konfigurationsdatei finden Sie nachfolgend:

```
<configuration>
   <system.web>
      <authentication>
         <forms>
            <credentials>
         <passport>
      <authorization>
         <allow>
         <deny>
      <browserCaps>
         <result>
         <use>
         <filter>
            <case>
      <clientTarget>
         <add>
         <remove>
         <clear>
      <compilation>
         <compilers>
            <compiler>
         <assemblies>
            <add>
            <remove>
            <clear>
      <customErrors>
         <error>
      <globalization>
      <httpHandlers>
         <add>
         <remove>
         <clear>
      <httpModules>
         <add>
         <remove>
         <clear>
      <httpRuntime>
      <identity>
      <machineKey>
      <pages>
```

```
      <processModel>
      <securityPolicy>
         <trustLevel>
      <sessionState>
      <trace>
      <trust>
      <webServices>
         <protocols>
            <add>
            <remove>
            <clear>
         <serviceDescriptionFormatExtensionTypes>
            <add>
            <remove>
            <clear>
         <soapExtensionTypes>
            <add>
         <soapExtensionReflectorTypes>
            <add>
         <soapExtensionImporterTypes>
            <add>
         <WsdlHelpGenerator>
      </webServices>
   </system.web>
</configuration>
```

Die wichtigsten Abschnitte werden nachfolgend vorgestellt. Einige Teile wirken jedoch auf ganz bestimmte Programmfunktionen und werden in den entsprechenden Abschnitten beschrieben:

- Das Element `<authentication>` wird in Abschnitt 10.5, »Sicherheit« ab Seite 972 beschrieben.
- Information zu `<webServices>` werden dagegen in Kapitel 13, »Webservices« ab Seite 1043 vorgestellt.
- Die Erstellung eigener HTTP-Handler und die Aktivierung im Abschnitt `<httpHandlers>` beschreibt Abschnitt 12.2, »HTTP-Handler« ab Seite 1024.
- Hinweise zur Konfiguration der Internationalisierung und Globalisierung finden Sie in Abschnitt 4.10.3, »Konfiguration in *web.config*« ab Seite 346. Verantwortlich dafür ist der Abschnitt `<globalization>`.
- Web Services dienen der Maschine-zu-Maschine-Kommunikation. Das Element `<webServices>` dient der Konfiguration. Erläuterungen dazu finden Sie im Kapitel 13, »Webservices« ab Seite 1043.

Alle anderen grundlegenden Einstellungen finden Sie nachfolgend.

10.3.1 Prinzipieller Umgang mit Konfigurationsdateien

Generell gelten die Einstellungen in der *web.config* für das Verzeichnis, in dem sie liegt. Das ist unter Umständen umständlich, wenn verschiedene Verzeichnisse einer Applikation unterschiedlich konfiguriert werden sollen. Es wäre sicher angenehm, alle Einstellungen in einer Datei zu verwalten.

Normalerweise stehen die applikationsspezifischen Parameter im Zweig `<system.web>`:

```
<configuration>
   <system.web>
      <!-- Hier stehen die Tags -->
   </system.web>
</configuration>
```

Einstellungen in web.config *für einen bestimmtes Verzeichnis werden immer automatisch an alle tieferliegenden vererbt. Durch die Option* `<location>` *können Sie dieses Verhalten modifizieren.*

Diese Einstellungen gelten für den Ort, an dem die Konfigurationsdatei selbst liegt.

Sie können der Datei nun weitere Abschnitte hinzufügen, `<location>` genannt:

```
<configuration>
   <system.web>
      <!-- Hier stehen die Tags -->
   </system.web>
   <location path="data/special">
       <system.web>
          <!-- Hier stehen die Tags -->
       </system.web>
    </location>
</configuration>
```

Ausgehend vom aktuellen Verzeichnis kann mit der Angabe `path` bestimmt werden, für welches Verzeichnis die enthaltenen Konfigurationsangaben gelten.

Die Einstellungen lassen sich natürlich auch für die gesamte Maschine vornehmen. Dies erfolgt dann in der Datei *machine.config*.

10.4 Konfiguration des gesamten Systems machine.config

Die Konfiguration eines Servers übernimmt die Datei *machine.config*. Fast alle Einstellungen, die hier erfolgen, lassen sich in *web.config* wieder überschreiben. Sie finden *machine.config* in folgendem Ordner:

%systemroot%\Microsoft.NET\Framework\v1.0.3705\CONFIG

Die Versionsnummer, unterstrichen dargestellt, muss gegebenenfalls an Ihre Installation angepasst werden.

10.4.1 Optionen des Compilers

ASP.NET-Codes wird mit einem Compiler übersetzt. Das passiert, wenn Sie das Framework SDK installiert haben, beim Aufruf der Seite automatisch. Wenn Sie Visual Studio .NET verwenden, kümmert sich die Entwicklungsumgebung um die nötigen Schalter und Einstellungen. Abweichungen lassen sich in den Eigenschaften eines Projekts einstellen.

Bei der automatischen Übersetzung bestehen solche Eingriffsmöglichkeiten über die *web.config*, basierend auf den in *machine.config* befindlichen Standardeinstellungen:

```
<compilation debug="false" explicit="true" defaultLanguage="vb">
  <compilers>
    <compiler language="c#;cs;csharp"
              extension=".cs"
              type="Microsoft.CSharp.CSharpCodeProvider, ↵
                    System, Version=1.0.3300.0, ↵
                    Culture=neutral, ↵
                    PublicKeyToken=b77a5c561934e089" ↵
              warningLevel="1" compilerOptions="/nostdlib"/>
    <compiler language="vb;vbs;visualbasic;vbscript"
              extension=".vb"
              type="Microsoft.VisualBasic.VBCodeProvider, ↵
                    System, Version=1.0.3300.0, ↵
                    Culture=neutral, ↵
                    PublicKeyToken=b77a5c561934e089"/>
```

```
    <compiler language="js;jscript;javascript"
              extension=".js"
              type="Microsoft.JScript.JScriptCodeProvider,↵
                    Microsoft.JScript, ↵
                    Version=7.0.3300.0, ↵
                    Culture=neutral, ↵
                    PublicKeyToken=b03f5f7f11d50a3a"/>
    </compilers>
</compilation>
```

Die Auflistung der bereit stehenden Compiler dürfte kaum Fragen aufwerfen. Anbieter von anderen Sprachen für .NET müssen ihre Software so gestalten, dass der Compiler und dessen Basisklasse hier vermerkt wird.

Interessanter sind die Attribute des Zweiges `compilation`>. Hier gibt es folgende Einstellmöglichkeiten:

- `debug="true|false"`

  Legt fest, ob der Compiler DEBUGGERCODE erzeugen soll oder nicht. Debuggercode führt im Code Marken mit, mit denen die Position der Haltepunkte in der Quelle festgestellt werden kann. Das macht den Code langsamer und größer, ist aber in der Entwicklungsphase notwendig. Die Option kann durch ein Attribut der Seitendirektive `@Page` überschrieben werden.

- `defaultLanguage="cs|vb|js"`

  Dies ist die Standardsprache, die in ASP.NET-Seiten für eingebetteten Code verwendet werden soll. Auch dieser Wert kann mit der Seitendirektive `@Page` überschrieben werden. Der Standardwert ist »vb«.

- `tempDirectory`

  Die temporären Dateien werden normalerweise in folgendem Pfad abgelegt:

  %systemroot%\Microsoft.NET\Framework\v1.0.3705\Temporary ASP.NET files

  Der unterstrichene Teil muss der installierten Version entsprechen. Temporäre Dateien entstehen, wenn der Compiler ASP.NET-Dateien übersetzt.

- `strict="true|false"`

  Schaltet die Strict-Option (`Option Strict`) in VB.NET ein oder aus. Standard ist »ein« (`true`).

- `explicit="true|false"`

  Schaltet die Explicit-Option (`Option Explicit`) in VB.NET ein oder aus. Standard ist »ein« (`true`).

- `batch="true|false"`

  Kontrolliert die Verfügbarkeit der Übersetzung von Stapelverarbeitungsdateien. Diese Option ist standardmäßig aktiv.

- `batchTimeout="15"`

  Wenn mehrere Übersetzungsvorgänge aus einer Stapelverarbeitungsdatei heraus erfolgen, begrenzt diese Option die Anzahl Sekunden, die der Vorgang in Anspruch nehmen darf. Der Standardwert beträgt 15 Sekunden.

- `maxBatchSize="1000"`

  Kontrolliert die maximale Zahl an Datenquellen, die in einem Batchlauf verwendet werden können. Der Standardwert beträgt 1.000.

- `maxBatchGeneratedFileSize="3000"`

  Bestimmt die maximale Dateigröße, die durch Übersetzung in einem Batchlauf entstehen darf. Der Standardwert beträgt 3.000 KByte.

- `numRecompilesBeforeAppRestart="15"`

  Anzahl der Compilerläufe, nach der die Applikation recycelt wird. Dies sichert die Zuverlässigkeit des Compilers.

In einem zweiten Unterelement `<assemblies>` werden außerdem die standardmäßig verfügbaren Assemblies definiert, die der Compiler verwenden darf. Hier sind die Klassen der Basisklassenbibliothek und einiger ergänzender Assemblies zu finden. Weitere, beispielsweise von Drittanbietern, können hinzugefügt werden.

Die Referenzen lassen sich aber auch im Visual Studio .NET hinzufügen, sodass die Option nur für reine SDK-Umgebungen interessant ist.

10.4.2 Den ASP.NET-Worker Process konfigurieren

ASP.NET läuft unabhängig vom IIS und kann deshalb weitgehend konfiguriert werden, auch wenn kein Zugriff auf die Metabasis (IIS 5) oder Konfigurationsdateien (IIS 6) besteht. Die Basis der ASP.NET-Engine bildet der so genannte »Worker Process« (der,

der die Arbeit macht). Realisiert ist dieses Programm als *aspnet_wp.exe*. Ob der Worker Process gestartet ist und wie viel Speicher er konsumiert, können Sie im Task-Manager sehen.

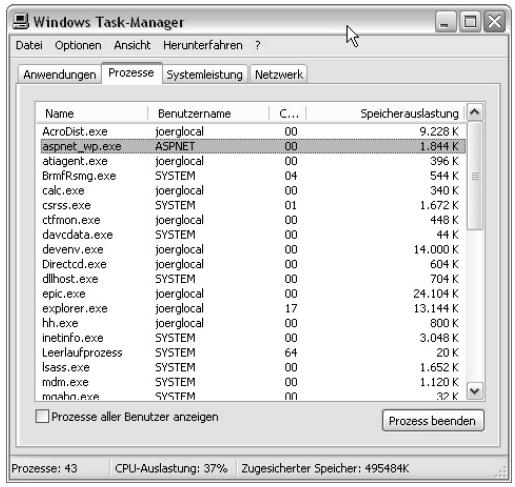


Abbildung 10.2: Der Worker Process im Task-Manager

Läuft der Worker Process, taucht er im Task-Manager auf. Im laufenden Betrieb eines Live-Servers kann so auch überwacht werden, welche Prozessorleistung allein die Verarbeitung der ASP.NET-Dateien in Anspruch nimmt. Es ist nahe liegend, dass hier Konfigurationsmöglichkeiten angebracht sind.

Einige Konfigurationseinstellungen des Worker Process betreffen auch den IIS. Sie werden nicht durch ASP.NET überwacht, sondern von aspnet_isapi.dll *beim Start des IIS gelesen. Nach allen Änderungen an dem nachfolgend beschriebenen Abschnitt* `<processModel>` *der Konfigurationsdatei* machine.config *muss der IIS neu gestartet werden.*

Den Worker Process konfigurieren: <processModel>

Die Konfiguration erfolgt im Tag `<processModel>`. Die einzelnen Attribute werden nachfolgend vorgestellt.

enable="true|false"

Grundsätzlich kann der Worker Process als externes Programm ein- und ausgeschaltet werden. »Ein« (`enable="true"`) ist der Standardwert. Wenn Sie die Option deaktivieren, wird der Worker Process nicht extern, sondern im IIS als In-Process-Applikation ge-

startet. Dies ist zwar schneller, aber auch unsicherer, weil ein Absturz oder Hängen des Worker Process den IIS mitreißt. Um die Änderung wirksam werden zu lassen, muss der WWW-PUBLISHINGDIENST gestoppt und gestartet werden.

timeout="Infinite|HH:MM:SS"

Das nächste Attribut `timeout` bestimmt die Zeit, die der Worker Process maximal mit der Abarbeitung einer Anforderung beschäftigt sein darf. Der Standardwert ist `Infinite` (unendlich). Wird die Zeit überschritten und läuft das Programm noch immer, startet ein neuer Worker Process. Falls Applikationen laufen, die Sie nicht ändern können und die nach einer gewissen Zeit immer mehr Speicher verbrauchen und damit Systemleistung kosten, können Sie den Zeitraum so einstellen, dass vor Eintritt des Engpasses ein neuer Prozess startet. Folgender Parameter startet den Worker Process alle 24 Stunden neu:

```
timeout="24:00:00"
```

idleTime="Infinite| HH:MM:SS"

Der Worker Process startet normalerweise bei der ersten eintreffenden Anforderung und bleibt dann in mindestens einer Instanz bestehen. Der Standardwert für das Attribut `idleTime` beträgt `Infinite`. Wird hier ein Zeitwert eingesetzt, beendet der Worker Process nach Ablauf der Zeit seine Arbeit und startet mit der nächsten Anforderung wieder. Der Neustart entspricht dem »recyceln« des Prozesses. Auch hier ist die Anwendung gegeben, wenn es Probleme mit steigendem Ressourcenverbrauch gibt.

shutDownTimeout="HH:MM:SS"

Der Worker Process hat normalerweise eine bestimmte Zeit zur Verfügung, um sich selbst zu beenden. Dieser Parameter – Standardwert fünf Sekunden – bestimmt diese Zeit. Möglicherweise laufen sehr anspruchsvolle Anwendungen, die längere Zeit brauchen, um Vorgänge abzuschließen und den Prozess später enden lassen. Dann kann die Zeit entsprechend erhöht werden. Wird der Zeitablauf erreicht, forciert ASP.NET das Ende des Prozesses ohne Rücksicht auf den Zustand laufender ASP.NET-Programme.

requestLimit="Infinite|Zahl"

Der Neustart des Worker Processes spielt generell eine große Rolle, um hoch belastete Server ständig mit maximaler Leistung betreiben zu können. Vor allem Provider, die viele ihnen unbekannte Applikationen laufen haben, profitieren von dieser Option. So kann mit folgendem Parameter der Neustart nach 500 Anforderungen forciert werden:

```
requestLimit="500"
```

Der Standardwert ist `Infinite`, es wird also normalerweise kein recyceln des Prozesses stattfinden.

requestQueue Limit="Zahl"

Ein anderen Fall tritt auch unter hoher Last ab und zu auf. Wenn zuviele Anforderungen anstehen, »stapelt« ASP.NET diese in einer Warteschlange. Nun kann das häufig passieren und darauf hindeuten, dass der Worker Process nicht mehr zufrieden stellend arbeitet. Standardmäßig dürfen 5.000 Anforderungen gestapelt werden. Ist diese Zahl erreicht, wird der Prozess recycelt.

memory Limit="Zahl"

Wenn einige wenige Prozesse zuviel Speicher in Anspruch nehmen, sollte der Abbruch nach dem Speicherverbrauch erfolgen. Als Abbruchkriterium wird hier nicht ein absoluter Wert benötigt, sondern eine Prozentangabe (ohne Prozentzeichen), die den Wert in Abhängigkeit vom physischen Speicher angibt:

```
memoryLimit="50"
```

Normalerweise sollten echte Speicherlecks in verwaltetem (managed) Code nicht auftreten. Wenn jedoch alte COM-Komponenten eingebunden werden, die natürlich nicht verwalteten Code enthalten, sind derartige Fehler möglich.

Neu in ASP.NET ist auch die explizite Unterstützung von Mehrprozessorsystemen. Microsoft bezeichnet dies allgemein als »Web Garden«. Mit Hilfe der Attribute `webGarden` und `cpuMask` wird festgelegt, wie viele Prozessoren verwendet werden.

userName= "Name" password="pw"

Der Worker Process läuft normalerweise unter dem Konto *ASPNET*. Sie können dies in Abbildung 10.2 in der Spalte *Benutzername* erkennen. Dieses Konto lässt sich ändern, indem die Attribute `userName` und `password` gesetzt werden.

logLevel="All| None|Error"

Fehler, die während der Arbeit des Worker Process erkannt werden, erscheinen im Ereignisprotokoll »Anwendung«. Ob nur Fehler oder all Ereignisse aufgezeichnet werden, kann mit dem Attribut `logLevel` entschieden werden. Die möglichen Optionen sind `All`, `None` und `Errors`. `Errors` ist der Standardwert.

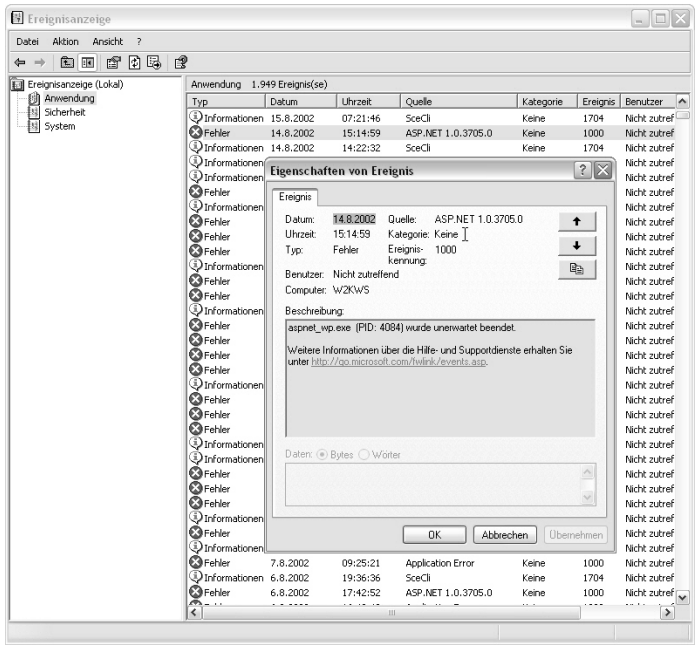


Abbildung 10.3: Fehler des Worker Process im Ereignisprotokoll

clientConnected-Check="Infinite|HH:MM:SS"

Mit einem weiteren Attribut kann das Verhalten des Clients überwacht werden. Wenn die Verbindung zwischen Webserver und Browser langsam ist, kann es vorkommen, dass Benutzer die Seite erneut anfordern, obwohl die vorhergehende Anforderung noch nicht beendet wurde. Dann stapeln sich die Anforderungen in der Anforderungswarteschlange. Standardmäßig prüft ASP.NET alle fünf Sekunden, ob neue Anforderungen des Clients vorliegen. Wenn das der Fall ist, wird nur die neueste bearbeitet, sodass die Arbeit des Prozesses entlastet wird, weil sinnlose Anforderungen, auf die ohnehin niemand mehr wartet, unbeantwortet bleiben.

Weitere Attribute sind speziellen Anwendungen vorbehalten und sollen hier nur überblicksweise gezeigt werden:

- Die Sicherheit von COM- und DCOM-Komponenten wird mit `comAuthenticationLevel` und `comImpersonate` geprüft.
- Wenn der seltene Fall eintritt, dass sich zwei Threads gegenseitig verriegeln (Deadlock), hilft `responseDeadlockInterval`. Dieses Attribut legt fest, wann dies erkannt wird; der Standardwert sind 3 Minuten (»00:03:00«). Mit `responseRestartDeadlockInterval` wird festgelegt, wie lange nach einem Abbruch

auf die nächste Prüfung gewartet wird (Standard: 9 Minuten (»00:09:00«).

- Zur Kontrolle der Threads des Worker Process dienen `maxWorkerThreads` (maximale Anzahl, Standardwert 25) und `maxIoThreads` (Ein-/Ausgabe-Threads, Standardwert ebenfalls 25).

10.5 Sicherheit

Dieser Abschnitt behandelt Sicherheit aus dem speziellen Blickwinkel der Web-Applikation. Eine mehr allgemein gehaltene Einführung in die Sicherheitsprinzipien des Frameworks finden Sie am Anfang des Buches im Abschnitt 1.6, »Allgemeine Sicherheit im Framework« ab Seite 85.

10.5.1 Das Sicherheitskonzept des IIS

ASP.NET nutzt den IIS als Schnittstelle zum Client. Der IIS mischt sich zwar nicht mehr in die eigentliche Verarbeitung von Seiten ein, ist aber als unterliegende Instanz Teil des Sicherheitssystems. Er verfügt über einige Sicherheitskonzepte, die für die Arbeit mit ASP.NET von Interesse sind:

- **Verschlüsselung der Kommunikation mit SSL (Secure Socket Layer)**

  Hiermit wird der Übertragungsweg geschützt. SSL verhindert wirkungsvoll, dass Dritte übertragene Daten lesen oder verändern können.

- **Beschränkung des Zugriffes auf bestimmte IP-Adressen oder Domänen**

  Durch die generelle Beschränkung des Zugriffes kann der Zugriff von geregelt werden, dass die Kommunikation mit unerwünschten Partnern gar nicht erst stattfindet. Dieses Verfahren ist für öffentliche Webserver kaum einsetzbar. Wenn Sie einen Webserver im Intranet installieren, ist der Einsatz sinnvoll.

- **Bereitstellung virtueller Verzeichnisse, die das Ausspähen der physischen Verzeichnisstruktur verhindern**

  Generell stellt der IIS ein virtuelles Verzeichnis für Ihre Applikation bereit. Wenn Sie Projekte mit Visual Studio .NET entwickeln, werden diese auch in einem solchen Verzeichnis installiert.

- **Prüfung von Zugriffsberechtigungen auf Basis der NTFS-Sicherheit**

  Über den IIS lassen sich mehrere Zugriffsmethoden einrichten, die eine Steuerung der Nutzung der Applikation auf Benutzerebenen erlauben. Damit haben Sie die genaueste Kontrolle, aber auch den höchsten Aufwand.

Wenn Sie ein Sicherheitskonzept für Ihre Applikation erarbeiten, können Sie diese vier Methoden einsetzen, um sich wirkungsvoll gegen Angriffe zu schützen.

Formen der Benutzerauthentifizierung

Der IIS lässt bestimmte Formen der Benutzerauthentifizierung zu, die je nach Einsatz und Sicherheitsbedarf verwendet werden können:

- **Anonymer Zugriff**

  Diese Form erlaubt den Zugriff ohne weitere Einschränkungen. Der IIS gestattet dem Client die Nutzung des Systemkontos *IUSR_MachineName*. Das gilt aber nur bis zum Aufruf des ASP.NET-Worker Process. Alle .NET-Instanzen werden unter dem ASP.NET-Konto *ASPNET* ausgeführt.

- **Authentifizierung mit Basic Authentication**

  Der Benutzer muss sich hier mit Benutzername und Kennwort identizifieren. Beide Angaben werden im Klartext übertragen – abgesehen von der üblichen Base64–Kodierung für binäre Inhalte. Das Verfahren muss verwendet werden, wenn der geschützte Zugriff beliebigen Clientprogrammen wie Netscape oder Opera gestattet werden soll.

- **Authentifizierung mit Digest Authentication**

  Dieses Verfahren erfordert auch die Angaben von Benutzername und Kennwort. Es wird aber nur ein Hashwert (MD5–verschlüsselt) übertragen, sodass ein Abfangen der Kennwortdaten nicht ausreicht, um daraus die Daten zu rekonstruieren. Auf Serverseite ist der Zugriff auf Active Directory erforderlich, als Client muss der Internet Explorer eingesetzt werden.

- **Integrierte Windows-Authentifizierung**

  Dieses Verfahren nutzt die interne Anmeldeprozedur von Windows und ist damit auf Windows-Clients beschränkt.

Auch hier wird ein Hashwert verwendet, sodass der Übertragungsweg sicher ist.

- **Sicherung über Clientzertifikate**

  Wenn bestimmte Benutzer sicher identifiziert werden sollen, können Clientzertifikate verwendet werden. ASP.NET stellt den Zugriff darauf über entsprechende Klassen bereit. Allerdings ist der Aufwand auf Benutzerseite sehr hoch, denn neben der kostenpflichtigen Beschaffung des Zertifikats ist auch die Installation im Browser erforderlich.

10.5.2 ASP.NET-Sicherheitsfunktionen im IIS

Für die Feststellung, welche Rechte ein konkreter Prozess im Rahmen einer Anforderung von Ressourcen nun hat, sind einige weitere Überlegungen notwendig. Die erste Instanz, mit der der Browser Kontakt aufnimmt, ist der IIS. Dieser läuft – als Programm *inetinfo.exe* – unter dem Konto *SYSTEM*. Dieses Konto hat natürlich sehr weitreichende Priviligien, sodass weitere Sicherheitsebenen wie Schalen darüberliegen. Die erste Ebene ist die so genannte Impersonifizierung des Prozesses. Der Benutzer hat sich immer in der einen oder anderen Weise identifiziert – dazu gehört auch die sichere Erkennung des Wunsches, anonym zu arbeiten. In diesem Fall wird das Konto *IUSR_MachineName* verwendet. Erfolgt ein Zugriff auf Komponenten (COM-Objekte), wird als Zugriffskonto *IWAM_MachineName* (WAM = Windows Application Management) verwendet. Die ASP.NET-Laufzeitumgebung ist davon ausgeschlossen, sie läuft immer unter dem Konto *ASPNET*, das bei der Installation eingerichtet wurde.

Die Authentifizierung in ASP.NET

ASP.NET bietet eine gute Unterstützung für verschiedene Authentifizierungarten an. Sie können je nach Bedarf entscheiden, welches System zum Einsatz kommt:

- **Windows-Authentifizierung**

  Siehe Abschnitt 10.5.5, »Windows-Authentifizierung« ab Seite 984.

- **Forms-Authentifizierung**

  Siehe Abschnitt 10.5.3, »Forms-Authentifizierung« ab Seite 976.

- **Passport**

  Bevor Sie sich daran machen, die Anmeldung über Passport zu realisieren, sollten Sie sich mit dem Konzept auseinander setzen. Passport ist ein Dienst der Microsoft .NET My Services-Palette und bietet Benutzern die Möglichkeit eines zentralen Login für alle teilnehmenden Webseiten. Das ist für Benutzer bequem und kostenlos. Für Unternehmen sieht das freilich anders aus. Sie müssen sich beim Passport-Dienst anmelden und den Schlüssel und Zugangscodes beschaffen, um daran teilnehmen zu können. Dies ist in der Praxis eine längere Prozedur. Nach der Anmeldung müssen Sie eine »Compliance«-Erklärung absenden und einem Tester bei Microsoft Zugang zu Ihrer Applikation verschaffen. Nach erfolgreichem Test wird Ihr Konto freigegeben und Sie nehmen nun mit Ihrer Site an Passport teil.

  Das Ganze macht Microsoft nicht aus Nächstenliebe. Da der Dienst für Benutzer kostenlos ist, stellt der Unternehmer die Einnahmequelle dar. Zu bezahlen sind derzeit einmalig $ 10.000 pro Unternehmen, auch wenn Sie mehrere Domänen anschließen. Weiterhin fallen $ 1.500 für jeden Compliance-Test an, also am Anfang mindestens ein Mal. Danach behält sich Microsoft weitere Tests zu je $ 1.500 vor, wobei jedoch auf einen jährlichen Rhythmus orientiert wird.

  Sie sollten sich deshalb genau überlegen, ob der Aufwand für Passport lohnt. Für eine Website mit hoher An- und Abmeldehäufigkeit, starkem Endkundenverkehr und möglicherweise weiteren kostenpflichtigen Diensten kann der Einsatz lohnenswert sein. Die Autoren haben Erfahrung in der Realisierung solcher Dienste und stehen interessierten Lesern hier gern im Rahmen einer Consultingleistung zur Verfügung.

Der anonyme Zugriff

Daneben besteht natürlich immer die Möglichkeit, auf eine explizite Authentifizierung zu verzichten. Das ist im übrigen auch die Standardeinstellung: Der anonyme Zugriff. Der Prozess läuft in diesem Fall unter dem Konto *ASPNET* ab, solange die Impersonifizierung deaktiviert bleibt[1]. Wenn die Impersonifizierung aktiviert wurde, übernimmt ASP.NET das Konto des IIS – im Falle des anonymen Zugriffs ist dies *IUSR_MachineName*.

1 Der deaktivierte Zustand entspricht der Standardeinstellung.

Um nun die gewünschte Zugriffsart festzulegen, nutzen Sie den Abschnitt `<authentication>` der Datei *web.config*.

```
<configuration>
  <system.web>
    <authentication mode="Windows" />
    ...
  </system.web>
</configuration>
```

Für das Attribut `mode` können Sie folgende Werte wählen: `Windows`, `Passport`, `Forms`, `None`. Innerhalb des Zweiges `<system.web>` können dann feinere Einstellungen vorgenommen werden, die in den entsprechenden Abschnitten weiter unten beschrieben werden.

Die Impersonifizierung in ASP.NET

Wenn sich nun aber ein Benutzer nicht anonym anmeldet, muss die Laufzeitumgebung in der Lage sein, ihm die Rechte einzuräumen, die er erwartet und die seinem Konto entsprechen. Dieser Vorgang wird als »Impersonifizierung« bezeichnet. Das heißt, der aktuelle Prozess nimmt die »Person« des Aufrufers an, um mit seinen Rechten operieren zu können.

Standardmäßig ist die Impersonifizierung abgeschaltet. Das heißt, dass trotz der Anmeldung über den IIS einem Benutzer nicht mehr Rechte eingeräumt werden, als das Konto *ASPNET* hergibt. Sie können diesen Vorgang explizit für den gesamten Server freigeben. Dazu öffnen Sie die Datei *machine.config* und suchen den Eintrag `<identity>`.

10.5.3 Forms-Authentifizierung

Die Forms-Authentifizierung unterstützt die Entwicklung einer eigenen Anmeldeprozedur basierend auf HTML-Formularen und Cookies. Das Sicherheitsniveau ist relativ niedrig, außerdem muss der Benutzer dauerhaft Cookies akzeptieren. Dafür ist der Einrichtungsaufwand gering. Die Datei *web.config* konfigurieren Sie für diese Funktion folgendermaßen:

```
<authentication mode="Forms" >
   <forms
      name="LogOnCookie"
      path="/"
```

```
      loginUrl="login.aspx"
      protection="All"
      timeout="25">
      <credentials passwordFormat="Clear">
         <user name="Joerg" password="haide" />
         <user name="Uwe" password="lukas" />
      </credentials>
   </forms>
</authentication>
<authorization>
  <deny users="?" />
</authorization>
```

Eine solche Konfiguration führt mehrere Schritte bei der Anforderung einer Ressource im betroffenen Verzeichnis aus. Zuerst wird geprüft, ob für den betreffenden Benutzer der Zugriff erlaubt oder verboten ist. Im Beispiel wird allen anonymen Benutzern der Zugriff untersagt:

```
<deny users="?" />
```

Sie können im Attribut `users` neben einer durch Kommata getrennten Listen von Namen auch ein Sternchen »« für »Alle« oder ein Fragezeichen »?« für »Anonym« angeben. Wenn Sie Domänennamen kontrollieren wollen, schreiben Sie »DOMAIN\Name«.*

Erst wenn ein solches Verbot greift, wird die Authentifizierungsprozedur gestartet. Erlauben Sie den Zugriff dagegen, wird auch keine Anmeldung angefordert.

Im Fall der Form-Authentifizierung wird ASP.NET nun Folgendes versuchen: Es wird ein Cookie erzeugt, basierend auf den Attributen des `<forms>`-Abschnittes. Dann wird die Ausführung an eine Anmeldeseite weitergeleitet, die im Beispiel den Namen *login.aspx* hat. Dies ist auch der Standardname, den ASP.NET verwendet, wenn Sie keinen angeben. Dort sollten Sie ein Formular programmieren, dass die benötigten Daten erfasst. Anschließend vergleichen Sie mit einem entsprechenden Programm, ob die Anmeldedaten Ihren Erwartungen entsprechen. Dann geben Sie die Anwendung frei. ASP.NET kümmert sich nun während der Sitzung um die laufende Identifizierung des Benutzers, basierend auf dem Cookie.

Beachten Sie, dass für die Attributnamen Groß- und Kleinschreibung entscheidend ist.

Die möglichen Einstellungen sind sehr variabel. Die folgende Tabelle zeigt die Bedeutung der Attribute des <forms>-Elements:

Attribut	Bedeutung und Optionen
name	Name des Cookies
loginUrl	URL, unter der das Anmeldeformular zu finden ist
protection	Verschlüsselungsart des Cookies: ▶ All – Cookie wird verschlüsselt und Daten werden überprüft. Dies ist die Standardeinstellung ▶ None – Keine Prüfung und Verschlüsselung ▶ Encryption – nur verschlüsseln ▶ Validation – Nur überprüfen
timeout	Laufzeit des Cookies in Minuten. Der Standardwert ist 30.
path	Pfad zur Anwendung, der dem Cookie zugewiesen wird. Der Standard ist »\« (Wurzel der Applikation)

Tabelle 10.4: Attribute für <forms>

Innerhalb des <forms>-Zweiges werden nun die speziellen Zugriffsbedingungen innerhalb des Zweiges <credentials> aufgeführt. Auch hierfür finden Sie nachfolgend das verfügbare Attribut:

Attribut	Bedeutung und Optionen
passwordFormat	Zeigt an, wie das Kennwort verschlüsselt ist: ▶ Clear – Keine Verschlüsselung ▶ MD5 – MD5-Hash ▶ SHA1 – SHA1–Hash Die Verschlüsselung zeigt an, dass der in der Datei befindliche Werte bereits in dieser Form kodiert ist.

Tabelle 10.5: Attribute für <credentials>

Bleibt zuletzt die Festlegung der Benutzer, Tag <user>, für die die Authentifzierung gilt. Auch hier sind einige Attribute:

Attribut	Bedeutung und Optionen
name	Der Anmeldename
password	Das erwartete Kennwort

Tabelle 10.6: Attribute für <user>

Die Nennung des Kennworts im Klartext in web.config *ist eine Sicherheitslücke, auch wenn ASP.NET den Zugriff auf diese Datei über das Web – egal in welchem Sicherheitskontext – niemals zulässt. Denn lokal besteht natürlich Zugriff darauf. Speichern Sie die Kennwörter daher nach Möglichkeit in der angegebenen Hash-Kodierung (MD5 oder SHA1).*

Programmierung einer Authentifizierungslösung

Die typische Programmierung einer Authentifizierungslösung besteht aus zwei Schritten:

1. Erstellen eines Eingabeformulars
2. Überprüfen der Eingabewerte und Reaktion darauf

Die Reaktion auf die Eingabe kann wiederum auf zwei Aktionen hinauslaufen: Freigabe der ursprünglich angeforderten Datei oder Ausgabe eines Hinweises über die misslungene Anmeldung. Beides ist sehr einfach zu programmieren. Das Formular *login.aspx* könnte folgendermaßen aussehen:

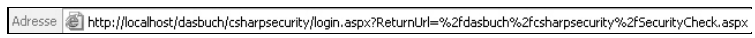
```
<h1>Anmeldung erforderlich</h1>
<form runat="server">
<table>
   <tr>
      <td>Name: </td>
      <td>
         <asp:TextBox Runat="server" ID="logName" />
      </td>
   </tr>
   <tr>
      <td>Kennwort: </td>
      <td>
         <asp:TextBox TextMode="Password" ↵
              Runat="server" ID="logPassword" />
      </td>
   </tr>
   <tr>
      <td></td>
      <td>
         <asp:Button Runat="server" ID="Submit" ↵
                     Text="Anmelden" ↵
                     OnClick="Login_Click"/>
      </td>
   </tr>
</table>
```

```
<asp:Label Runat="server" ID="LoginError"
           BackColor="Red" ForeColor="White"/>
</form>
```

Listing 10.1: Ein Anmeldeformular (login.aspx)

Ablauf der Authentifizierung

Wenn nun eine geschützte Datei angefordert wird, leitet ASP.NET zu diesem Formular um. Sie erkennen dies durch die Veränderung des URL:



Dort erhält der Benutzer die Möglichkeit, die Daten einzugeben. Wird auf die Schaltfläche Anmelden geklickt, wird die entsprechende Ereignisbehandlungsmethode ausgeführt.

Abbildung 10.4: Das Anmeldeformular nach einer fehlerhaften Anmeldung

Der Aufwand ist offensichtlich überschaubar, wie das folgende Listing zeigt:

```
public class login : System.Web.UI.Page
{
   protected TextBox logName;
   protected TextBox logPassword;
   protected Label LoginError;

   public void Login_Click (object sender, EventArgs e)
   {
      if (FormsAuthentication.Authenticate(logName.Text,↵
          logPassword.Text))
      {
         FormsAuthentication.RedirectFromLoginPage↵
                              (logName.Text, true);
      }
      else
      {
         LoginError.Text = "Fehler: Anmeldedaten falsch.↵
```

```
Beachten Sie Groß- und Kleinschreibung";
      }
   }

}
```

Listing 10.2: Auswertung der Eingabe und Prüfen der Anmeldedaten (login.aspx.cs)

Wie es funktioniert

Um Zugriff auf die Klassen zur Auswertung der Anmeldeinformationen zu erlangen, müssen Sie noch den Namensraum `System.Web.Security` einbinden:

```
using System.Web.Security;
```

Im eigentlichen Programm prüfen Sie nun, ob Name und Kennwort denen in der *web.config* entsprechen:

```
if (FormsAuthentication.Authenticate
   (logName.Text, logPassword.Text))
```

Wenn das der Fall ist, lösen Sie eine Weiterleitung auf die ursprünglich aufgerufene Seite aus:

```
FormsAuthentication.RedirectFromLoginPage(logName.Text, true);
```

Tipp: permanente Cookies

Diese statische Methode bietet als zweiten Parameter die Möglichkeit, dass mit der Weiterleitung gesendete Authentifizierungs-Cookie permanent zu speichern. Diese Wahl können Sie – im Sinne einer guten Benutzerführung – auch dem Benutzer überlassen. Dazu könnten Sie ein `CheckBox`-Steuerelement auf der Anmeldeseite platzieren. Wird es aktiviert, wird der Parameter gleich `true` gesetzt. Dann wird das Ablaufdatum nicht auf die in der *web.config* definierten Minuten (im Beispiel waren es 25) gesetzt, sondern soweit in die Zukunft, dass es praktisch permanent wird.

Es ist schlechter Stil, permanente Cookies ohne Einverständnis des Benutzers zu speichern. Mit der offenen Bekundung der Verwendung eines Cookies und der Wahlmöglichkeit durch den Benutzer steigt die Nutzungsrate deutlich.

Der *else*-Zweig behandelt nun noch den Fall einer fehlerhaften Anmeldung.

Weitere Methoden der Klasse FormsAuthentication

Verwendet wurden bereits die Methode `Authenticate` (Prüfung der Anmeldeparameter) und `RedirectFromLoginPage` (Setzen des

Cookies und Ausführen der Weiterleitung). Sie können diesen Prozess auch feiner steuern, wenn Sie die weiteren Methoden kennen.

Ebenfalls statisch ist `GetRedirectUrl`. Beim Aufruf wird der URL zurückgegeben, der zur Weiterleitung verwendet wird. Sie können den Wert nicht ändern. Allerdings ist es natürlich möglich, eine eigene Weiterleitung mit `Response.Redirect` (extern) oder `Server.Transfer` (intern) auszuführen und auf die Dienste von `RedirectFromLoginPage` zu verzichten.

Allerdings müssten Sie dann auch das Cookie selbst setzen. Dazu kommt die statische Methode `GetAuthCookie` in Frage. Hiermit wird ein Cookie-Objekt vom Typ `HttpCookie` erzeugt. Als Parameter ist der Anmeldename des Benutzers und das Ablaufverhalten anzugeben. Das Ablaufverhalten wird als zweiter Parameter übergeben – `true` für ein permanentes Cookie, `false` für eines mit der definierten Laufzeit. Sie können das Cookie nun verändern und dann mit `SetAuthCookie` wieder setzen, sodass es von `RedirectFromLoginPage` verarbeitet werden kann.

Mehr zu Cookies

Mehr allgemeine Informationen zu Cookies finden Sie in Abschnitt 8.5, »Cookies« ab Seite 795.

SignOut: Abmelden

Bleibt als letzte Maßnahme einer vernünftigen Benutzeranmeldung noch eine Abmeldemöglichkeit. mit Hilfe der Methode `SignOut`, die keine Parameter verlangt, wird das Cookie zerstört und damit bei der nächsten Weiterleitung vom Browser entfernt. Ruft der Benutzer danach erneut die Seite auf, wird die Anmeldeprozedur erneut starten. Auch eine designierte Abmeldeoption gehört zu einer guten Benutzerführung.

10.5.4 Personalisierung

Ist die Authentifizierung gelungen, stehen weit mehr Möglichkeiten zur Verfügung, als eine einfache Aussage über die Zulassung eines Kontos. So ist es sicher sinnvoll, einem Benutzer nach erfolgter Anmeldung persönliche Dienste bereit zu stellen. Dies kann auf jeder Seite anders aussehen. Sie benötigen also einen permanenten Zugriff auf die Authentifizierungsdaten innerhalb Ihrer Programme.

Der Benutzer als Objekt: User

Der Benutzer, ist er erstmal erkannt worden, steht dem Programmierer als Objekt `User` zur Verfügung. Zugriff darauf erhalten Sie über `HttpContext`. Diese Klasse stellt allgemein alle Informationen über den Kontext der aktuellen Anforderung und Applikation dar. Der Kontext umfasst im Objektmodell der Seite Informationen über Anforderung und Antwort, die Sitzung, den Benutzer und die Seite mit den Steuerelementen selbst.

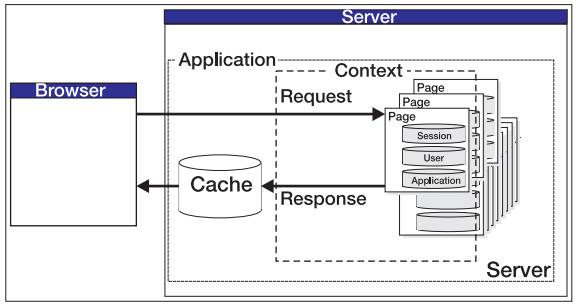


Abbildung 10.5: Einordnung der `HttpContext`-Klasse in das Objektmodell

Der Zugriff ist relativ einfach. Das folgende Beispiel zeigt eine Möglichkeit. In der *asxp*-Seite gibt es ein `Label`-Steuerelement, dass hier zur Ausgabe verwendet wird:

```
private void Page_Load(object sender, System.EventArgs e)
{
   IIdentity user = HttpContext.Current.User.Identity;

   if (user.IsAuthenticated)
   {
      Footer.Text = "Willkommen, " + user.Name;
   }
   else
   {
      Footer.Text = "Oops. Noch unbekannt?";
   }
}
```

Listing 10.3: Zugriff auf den authentifizierten Benutzer (Ausschnitt aus SecurityCheck.aspx.cs)

Wie es funktioniert

Mit `IsAuthenticated` wird hier sicherheitshalber festgestellt, ob die Authentifizierung bereits erfolgte. Wenn Sie das letzte Beispiel mit der Forms-Authentifizierung aktiviert und verwendet haben,

sollte das der Fall sein. Andernfalls erscheint die Fehlermeldung. Über die Eigenschaft `Name` können Sie auf den Anmeldenamen zugreifen. Es ist damit ein leichtes, persönliche Informationen aus einer Datenbank zu holen und die Anzeige der Seite danach zu modifizieren.

Statt mit Formularen können Sie Benutzer auch über ein reguläres Konto in der lokalen Benutzerverwaltung oder im Active Directory identifizieren. Dies dürfte jedoch nur im Intranet praktikabel sein. Lesen Sie im nächsten Abschnitt, wie es funktioniert.

10.5.5 Windows-Authentifizierung

Die Windows-Authentifizierung basiert auf der Prüfung der Anmeldung mit Hilfe der NT-Benutzerverwaltung bzw. des Active Directory. Zusätzlich zur Aktivierung der Windows-Authentifizierung selbst muss die Impersonifizierung aktiviert werden, denn nur so gelangt der Anmeldedatenstrom über den IIS bis zu ASP.NET vor.

Für den Benutzer gilt, dass bei dieser Form der Authentifizierung das Standarddialogfeld des Browsers verwendet wird. Gestalterische Freiheiten wie bei der Forms-Authentifizierung gibt es hier nicht.

Die Ressourcen müssen Sie, wenn Sie Windows die Kontrolle überlassen, natürlich auch über NTFS schützen. Dazu rufen Sie für den betreffenden Dateien und Verzeichnisse den Eigenschaften-Dialog auf und tragen dort auf der Registerkarte SICHERHEIT die Benutzer oder Gruppen ein, denen der Zugriff gewährt werden soll.

Die Windows-Authentifizierung einrichten

Zur Aktivierung modifizieren Sie die Datei *web.config* folgendermaßen:

```
<authentication mode="Windows" />
<identity impersonate="true" />
<authorization>
   <deny users="?" />
</authorization>
```