Anhang: Die technischen Grundlagen

Dieser Abschnitt bietet weiteres Hintergrundwissen und erläutert im Detail, wie bestimmte Konzepte und Prozesse funktionieren, die im Hauptteil des Handbuchs besprochen wurden.

Verarbeitung in IIS und ASP.NET

Hinweis: Die Informationen in diesem Abschnitt beziehen sich auf die Internet-Informationsdienste (IIS) 5 unter Windows 2000.

Der Code von ASP.NET-Webanwendungen und –Webdiensten wird in einer einzigen Instanz des ASP.NET-Workerprozesses (**Aspnet_wp.exe**) ausgeführt, wobei auf Multiprozessorcomputern auch mehrere Instanzen, d. h. eine pro Prozessor, konfiguriert werden können.

IIS authentifiziert Aufrufer und erstellt ein Windows-Zugriffstoken für den Aufrufer. Wenn in IIS die anonyme Authentifizierung aktiviert ist, erstellt IIS ein Windows-Zugriffstoken für das anonyme Internetbenutzerkonto (standardmäßig IUSR_MACHINE).

Anforderungen für ASP.NET-Dateitypen werden von einer ASP.NET-ISAPI-Erweiterung (**Aspnet_isapi.dll**) verarbeitet, die im IIS-Prozessadressraum (**Inetinfo.exe**) ausgeführt wird Hierbei wird eine Named Pipe für die Kommunikation mit dem ASP.NET-Workerprozess verwendet, wie in Abbildung 1 dargestellt. IIS übergibt das Windows-Zugriffstoken, das den Aufrufer repräsentiert, an den ASP.NET-Workerprozess. Das Windows-Authentifizierungsmodul von ASP.NET greift hierauf zurück, um ein **WindowsPrincipal**-Objekt zu erstellen, und das ASP.NET-Dateiautorisierungsmodul führt auf dieser Basis Windows-Zugriffsprüfungen durch, um sicherzustellen, dass der Aufrufer über die Berechtigung zum Zugriff auf die Datei verfügt.

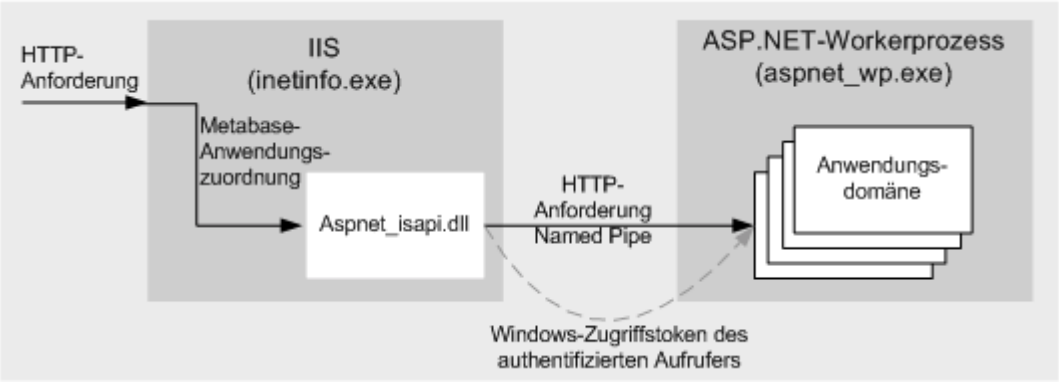

Abbildung 1
Kommunikation zwischen IIS und ASP.NET

Hinweis: Zugriffstoken sind prozessbezogen. Dementsprechend ruft die ASP.NET-ISAPI-DLL, die in **Inetinfo.exe** ausgeführt wird, **DuplicateHandle** auf, um das Tokenhandle in den Prozessadressraum von **Aspnet_wp.exe** zu duplizieren. Anschließend wird der Handlewert durch die Named Pipe weitergeleitet.

Anwendungsisolation

Innerhalb des Workerprozesses werden separate Anwendungsdomänen (eine pro virtuellem IIS-Verzeichnis oder, anders ausgedrückt, eine pro ASP.NET-Webanwendung oder –Webservice) verwendet, um Isolation zu gewährleisten.

Dies steht im Gegensatz zu den klassischen ASPs, bei denen die innerhalb der IIS-Metabasis konfigurierte Anwendungsschutzebene festlegt, ob die ASP-Anwendung "in-process" mit IIS (**Inetinfo.exe**), "out-of-process" in einer dedizierten Instanz von **Dllhost.exe** oder in einer gemeinsamen (gepoolten) Instanz von **Dllhost.exe** ausgeführt werden soll.

Wichtig: Die Einstellung für die Prozessisolationsebene in IIS hat keine Auswirkungen auf die Art und Weise, wie ASP.NET-Webanwendungen verarbeitet werden.

Die ISAPI-Erweiterung von ASP.NET

Die ISAPI-Erweiterung von ASP.NET (**Aspnet_isapi.dll**) wird im IIS-Prozessadressraum (**Inetinfo.exe**) ausgeführt und leitet Anforderungen betreffend ASP.NET-Dateitypen durch eine Named Pipe an den ASP.NET-Workerprozess weiter.

Bestimmte ASP.NET-Dateitypen werden mithilfe von Zuordnungen, die in der IIS-Metabasis definiert sind, ASP.NET-ISAPI-Erweiterungen zugeordnet. Zuordnungen für standardmäßige ASP.NET-Dateitypen (einschließlich ASPX, ASMX, REM und SOAP) werden bereits bei der Installation des .NET Frameworks hergestellt.

▶ **So zeigen Sie Anwendungszuordnungen an**

1. Starten Sie die Internet-Informationsdienste aus der Programmgruppe **Verwaltung**.
2. Klicken Sie mit der rechten Maustaste auf die Standardwebsite auf dem Webservercomputer, und klicken Sie dann auf **Eigenschaften**.
3. Klicken Sie auf die Registerkarte **Basisverzeichnis** und dann auf **Konfiguration**.
   Nun wird eine Liste der Zuordnungen angezeigt. Dieser Liste können Sie entnehmen, welche Dateitypen **Aspnet_isapi.dll** zugeordnet sind.

IIS 6.0 und Windows Server 2003

Mit IIS 6.0 unter Windows Server 2003 werden einige grundlegende Änderungen an der aktuellen Prozessanordnung eingeführt.

- Sie sind nun in der Lage, mehrere Anwendungspools zu konfigurieren, von denen jeder von einer oder mehreren Prozessinstanzen (**W3wp.exe**) bedient wird. Auf diese Weise ergeben sich zum einen Vorteile bei der Fehlertoleranz und der Verwaltbarkeit, und zum anderen können Sie separate Anwendungen in separaten Prozessen isolieren.
- ASP.NET ist in den Kernelmodus-HTTP-Listener von IIS 6.0 integriert, wodurch es möglich ist, Anfragen direkt vom Betriebssystem an den ASP.NET-Workerprozess zu übergeben.

Weitere Informationen

Weitere Informationen über IIS 6 finden Sie im TechNet im Artikel "IIS 6 Overview" (http://www.microsoft.com/technet/treeview/default.asp?url=/TechNet/prodtechnol/iis/evaluate/iis6ovw.asp, englischsprachig).

Pipelineverarbeitung in ASP.NET

Die Authentifizierungs- und Autorisierungsmechanismen von ASP.NET werden unter Verwendung von HTTP-Modulobjekten implementiert, die als Teil der standardmäßigen Pipelineverarbeitung von ASP.NET aufgerufen werden. Einzelne Webanforderungen und Antworten werden durch eine Pipeline aus Objekten geleitet, wie in Abbildung 2 veranschaulicht.

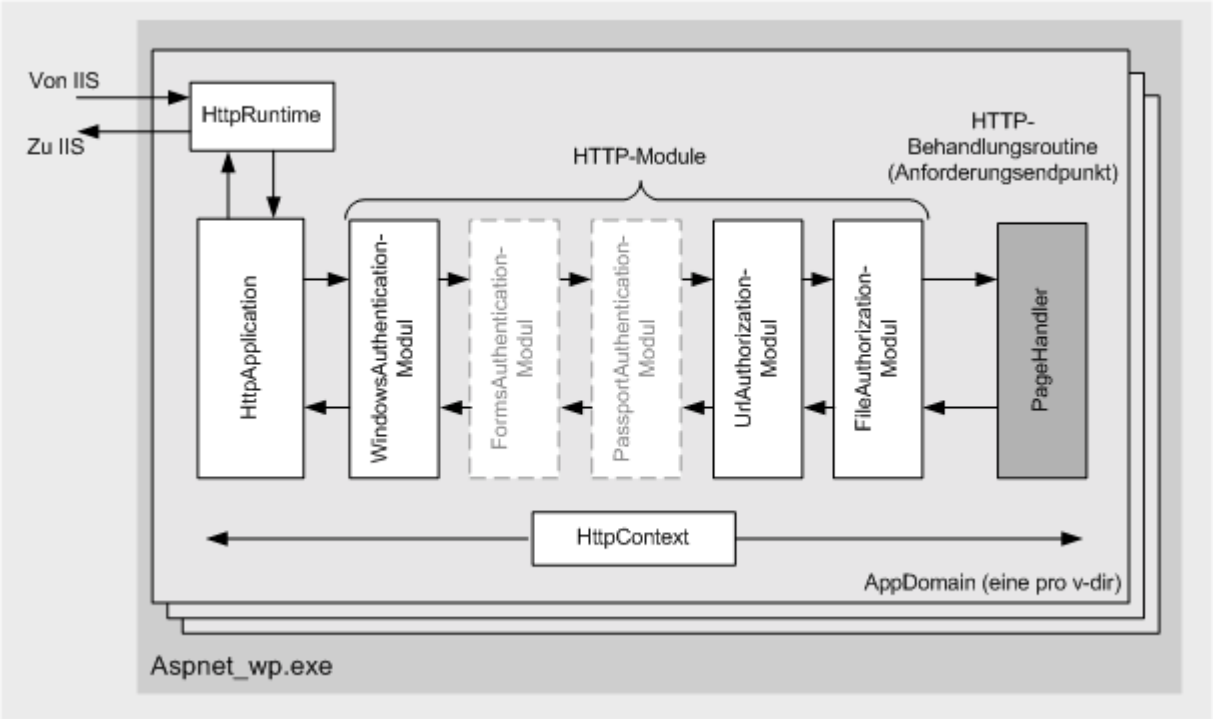

Abbildung 2
Pipelineverarbeitung in ASP.NET

Das Pipelinemodell von ASP.NET setzt sich aus einem **HttpApplication**-Objekt, unterschiedlichen HTTP-Modulobjekten sowie einem HTTP-Handlerobjekt mit den jeweils zugehörigen Factory-Objekten zusammen, die in Abbildung 2 zwecks besserer Übersichtlichkeit weggelassen wurden. Zu Beginn der Verarbeitungssequenz wird ein **HttpRuntime**-Objekt verwendet, und ein **HttpContext**-Objekt wird über die gesamte Lebensdauer der Anforderung eingesetzt, um Einzelheiten über Anforderung und Antwort zu übermitteln.

Die folgende Liste erläutert die Zuständigkeiten und Funktionen der mit der HTTP-Verarbeitungspipeline zusammenhängenden Objekte:

- Das **HttpRuntime**-Objekt prüft die von IIS empfangene Anforderung und gibt diese zwecks Verarbeitung an eine geeignete Instanz des **HttpApplication**-Objekts weiter. In jeder Anwendungsdomäne in **Aspnet_wp.exe** gibt es einen Pool aus **HttpApplication-Objekten**. Zwischen Anwendungsdomänen, **HttpApplication**-Objekten und den virtuellen IIS-Verzeichnissen besteht eine 1:1-Zuordnung. Mit anderen Worten, ASP.NET behandelt separate virtuelle IIS-Verzeichnisse auch als separate Anwendungen.

  Hinweis: In jeder Webanwendungsdomäne gibt es eine Instanz von **HttpRuntime**.

- Die **HttpApplication**-Objekte steuern die Pipelineverarbeitung. Für die Verarbeitung jeder simultan eingehenden HTTP-Anforderung wird ein einzelnes **HttpApplication**-Objekt erstellt. **HttpApplication**-Objekte werden aus Gründen der Leistungsfähigkeit gepoolt.

- HTTP-Modulobjekte sind Filter, die HTTP-Anforderungen und -Antwortnachrichten auf ihrem Weg durch die Pipeline verarbeiten. HTTP-Modulobjekte können den Inhalt der Anforderungs- und Antwortnachrichten einsehen oder ändern. HTTP-Module sind Klassen, mit denen **IHttpModule** implementiert wird.
- HTTP-Handlerobjekte sind die Endpunkte von HTTP-Anforderungen. Sie übernehmen die Anforderungsverarbeitung für bestimmte Dateitypen. So verarbeitet beispielsweise ein Handler Anforderungen für ASPX-Dateien, während ein anderer Anforderungen für ASMX-Dateien verarbeitet. Die HTTP-Antwortnachricht wird erzeugt und vom HTTP-Handler zurückgegeben. HTTP-Handler sind Klassen, mit denen **IHttpHandler** implementiert wird.
- Ein **HttpContext**-Objekt wird überall in der Pipeline verwendet, denn es repräsentiert die aktuelle Webanforderung und -antwort. Dieses Objekt steht für alle Module in der Pipeline und für das Handlerobjekt am Ende der Pipeline zur Verfügung. Das **HttpContext**-Objekt stellt verschiedene Eigenschaften einschließlich der **User**-Eigenschaft bereit, welche das **IPrincipal**-Objekt enthält, das wiederum den Aufrufer repräsentiert.

Die Anatomie einer Webanforderung

Die ASP.NET ISAPI-Bibliothek (**Aspnet_isapi.dll**) wird im Prozessadressraum von IIS (**Inetinfo.exe**) ausgeführt. Hiermit werden Anforderungen an das **HttpRuntime**-Objekt im ASP.NET-Workerprozess weitergegeben. Die nachstehenden Aktionen erfolgen in Reaktion auf jede Webanforderung, die von ASP.NET empfangen wird:

- Das **HttpRuntime-Objekt** prüft die Anforderung und übermittelt sie an eine Instanz des **HttpApplication**-Objekts.

  Es gibt mindestens ein **HttpApplication**-Objekt pro Anwendungsdomäne (die Objekte werden gepoolt) und eine Anwendungsdomäne pro virtuellem IIS-Verzeichnis. Die erste Anforderung für eine Datei in einem speziellen virtuellen Verzeichnis bewirkt, dass eine neue Anwendungsdomäne und ein neues **HttpApplication**-Objekt erstellt wird.
- Aus **Machine.config** wird eine Liste der HTTP-Module ausgelesen (die sich im <**httpModules**>-Element befinden). Der **Web.config** können weitere benutzerdefinierte HTTP-Module für eine bestimmte Anwendung hinzugefügt werden. Das standardmäßige <**httpModules**>-Element in **Machine.config** können Sie dem folgenden Codeausschnitt entnehmen.

```
<httpModules>
  <add name="OutputCache"
       type="System.Web.Caching.OutputCacheModule"/>
  <add name="Session"
       type="System.Web.SessionState.SessionStateModule"/>
  <add name="WindowsAuthentication"
       type="System.Web.Security.WindowsAuthenticationModule"/>
  <add name="FormsAuthentication"
       type="System.Web.Security.FormsAuthenticationModule"/>
  <add name="PassportAuthentication"
       type="System.Web.Security.PassportAuthenticationModule"/>
  <add name="UrlAuthorization"
       type="System.Web.Security.UrlAuthorizationModule"/>
  <add name="FileAuthorization"
       type="System.Web.Security.FileAuthorizationModule"/>
</httpModules>
```

  Die Authentifizierungsmodule binden das **AuthenticateRequest**-Ereignis ein, während die Autorisierungsmodule das **AuthorizeRequest**-Ereignis einbindet.

  Die Anforderung durchläuft jedes Modul in der Pipeline, obwohl nur ein einziges Authentifizierungsmodul geladen ist. Dies hängt von der Konfiguration des

<**authentication**>-Elements in **Web.config** ab. Beispielsweise sorgt das folgende <**authentication**>-Element dafür, dass **WindowsAuthenticationModule** geladen wird.

```
<authentication mode="Windows" />
```

- Das aktivierte Authentifizierungsmodul ist für die Erstellung eines **IPrincipal**-Objekts und für dessen Speicherung in der **HttpContext.User**-Eigenschaft zuständig. Dies ist von ausschlaggebender Bedeutung, da die nachfolgenden Autorisierungsmodule dieses **IPrincipal**-Objekt als Grundlage für die jeweilige Autorisierungsentscheidung verwenden.

  Bei fehlender Authentifizierung (wenn in IIS beispielsweise der anonyme Zugriff möglich und ASP.NET mit **<authentication mode ="None" />** konfiguriert ist) gibt es ein spezielles, nicht konfiguriertes Modul, das einen standardmäßigen anonymen Principal in die **HttpContext.User**-Eigenschaft setzt. Dementsprechend ist **HttpContext.User** nach der Authentifizierung immer ungleich Null.

  Wenn Sie ein benutzerdefiniertes Authentifizierungsmodul implementieren, muss der Code im benutzerdefinierten Modul ein **IPrincipal**-Objekt erstellen und dieses in **HttpContext.User** speichern.

Hinweis: Nach dem **AuthenticateRequest**-Ereignis verknüpft ASP.NET auch **Thread.CurrentPrincipal** basierend auf **HttpContext.User**.

- **HttpApplication** löst das **AuthenticateRequest**-Ereignis aus, welches in **Global.asax** eingebunden werden kann. Dies ermöglicht Ihnen, benutzerdefinierten Verarbeitungscode einzufügen, beispielsweise, um die mit dem aktuellen Benutzer verbundene Rollenliste zu laden. Beachten Sie jedoch, dass dies von **WindowsAuthenticationModule** automatisch erledigt wird. Die Rollenliste wird aus der Reihe der Windows-Gruppen ausgelesen, denen der authentifizierte Windows-Benutzer angehört.
- Nachdem das jeweils geeignete Authentifizierungsmodul seine Arbeit beendet hat, werden, sofern die Anforderung nicht zurückgewiesen wurde, die Autorisierungsmodule aufgerufen.
- Wenn **UrlAuthorizationModule** aufgerufen wird, prüft dieses **Machine.config** und **Web.config** auf Vorhandensein eines <**authorization**>-Tags. Ist ein solches Tag vorhanden, wird das **IPrincipal**-Objekt aus **HttpContext.User** abgerufen und geprüft, ob der Benutzer über die Berechtigung zum Zugriff auf die angeforderte Ressource unter Verwendung des angegebenen Verbs (GET, POST usw.) verfügt.

  Verfügt der Benutzer nicht über die adäquate Berechtigung, ruft **UrlAuthorizationModule** die Methode **HttpApplication.CompleteRequest** auf, wodurch die normale Nachrichtenverarbeitung abgebrochen wird. **UrlAuthorizationModule** gibt in diesem Fall einen Statuscode HTTP 401 zurück.
- Als nächstes wird **FileAuthorizationModule** aufgerufen. Hiermit wird geprüft, ob das **IIdentity**-Objekt in **HttpContext.User.Identity** eine Instanz der **WindowsIdentity**-Klasse ist.

  Wenn **IIdentity** kein Bestandteil von **WindowsIdentity** ist, stellt **FileAuthorizationModule** jede weitere Verarbeitung ein.

  Andernfalls ruft **FileAuthorizationModule** (über **P/Invoke**) die **AccessCheck**-API auf, um zu prüfen, ob der authentifizierte Aufrufer (dessen Zugriffstoken von IIS an ASP.NET übergeben wurde und vom **WindowsIdentity**-Objekt bereitgestellt wird) über die Berechtigung zum Zugriff auf die angeforderte Datei verfügt. Wenn der Dateisicherheitsdeskriptor zumindest über einen ACE (Access Control Entry oder Zugriffssteuerungseintrag) zum Lesen in seiner DACL (Discretionary Access List oder Zugriffssteuerungsliste) verfügt, darf die Anforderung weiterverarbeitet werden. Andernfalls ruft **FileAuthorizationModule** die Methode **HttpApplication.CompleteRequest** auf und gibt Statuscode 401 zurück.

Prozesse bei der Formularauthentifizierung

Das **FormsAuthenticationModule** wird aktiviert, wenn sich in **Web.config** das folgende Element befindet.

```
<authentication mode="Forms" />
```

Denken Sie daran, dass Sie für die Formularauthentifizierung das **Application_Authenticate**-Ereignis in **Global.asax** implementieren müssen. Die Formularauthentifizierung läuft wie folgt ab:

- Innerhalb dieses Codes können Sie ein **IPrincipal**-Objekt erzeugen und in **HttpContext.User** speichern. Dieses Objekt umfasst in der Regel die aus einem benutzerdefinierten Datenspeicher (normalerweise eine SQL Server-Datenbank oder Active Directory) abgerufene Rollenliste. Beim **IPrincipal**-Objekt handelt es sich gemeinhin um eine Instanz der **GenericPrincipal**-Klasse, es kann jedoch auch eine benutzerdefinierte **IPrincipal**-Klasse verwendet werden.

  FormAuthenticationModule prüft nun, ob ein **IPrincipal**-Objekt erzeugt wurde. Falls ja, wird dieses Objekt von den nachgeordneten Autorisierungsmodulen verwendet. Falls nicht, erzeugt **FormsAuthenticationModule** ein **GenericPrincipal**-Objekt (ohne Rollen) und speichert dieses im Kontext.

  Wenn keine Rolleninformationen vorhanden sind, schlagen sämtliche Autorisierungsprüfungen (wie z. B. PrincipalPermission-Forderungen) fehl.
- **UrlAuthorizationModule** verarbeitet das **AuthorizeRequest**-Ereignis. Die Autorisierungsentscheidung basiert auf dem **IPrincipal**-Objekt, das in **HttpContext.User** enthalten ist.

Prozesse bei der Windows-Authentifizierung

WindowsAuthenticationModule wird aktiviert, wenn sich in **Web.config** das folgende Element befindet.

```
<authentication mode="Windows" />
```

Die Windows-Authentifizierung läuft folgendermaßen ab:

1. **WindowsAuthenticationModule** erzeugt ein **WindowsPrincipal**-Objekt unter Verwendung des Windows-Zugriffstokens, das von IIS an ASP.NET übergeben wurde.
2. Mithilfe von **P/Invoke** werden Win32-Funktionen aufgerufen, um die Liste der Windows-Gruppen abzurufen, denen der Benutzer angehört. Diese werden zum Füllen der **WindowsPrincipal**-Rollenliste verwendet.
3. Das **WindowsPrincipal**-Objekt wird in **HttpContext.User** gespeichert und kann dann von den nachgeordneten Autorisierungsmodulen verwendet werden.

Ereignisbehandlung

Das **HttpApplication**-Objekt löst die Reihe der in Tabelle 1 aufgeführten Ereignisse aus. Einzelne HTTP-Module können diese Ereignisse einbinden (indem eigene Ereignishandler bereitgestellt werden).

Tabelle 1: Vom **HttpApplication**-Objekt ausgelöste Ereignisse

Ereignis	Hinweise
BeginRequest	Wird vor Beginn der Anforderungsverarbeitung ausgelöst
AuthenticateRequest	Authentifiziert den Aufrufer
AuthorizeRequest	Führt Zugriffsprüfungen durch
ResolveRequestCache	Ruft eine Antwort aus dem Cache ab
AcquireRequestState	Lädt den Sitzungsstatus
PreRequestHandlerExecute	Wird unmittelbar vor der Übergabe der Anforderung an das Handlerobjekt ausgelöst
PostRequestHandlerExecute	Wird unmittelbar nach der Übergabe der Anforderung an das Handlerobjekt ausgelöst
ReleaseRequestState	Speichert den Sitzungsstatus
UpdateRequestCache	Aktualisiert den Antwortcache
EndRequest	Wird nach Abschluss der Verarbeitung ausgelöst
PreSendRequestHeaders	Wird ausgelöst, bevor Antwortheader gesendet werden
PreSendRequestContent	Wird ausgelöst, bevor der Antworthauptteil gesendet wird

Hinweis: Der HTTP-Handler wird zwischen den Ereignissen **PreRequestHandlerExecute** und **PostRequestHandlerExecute** ausgeführt.

Die letzten beiden Ereignisse sind nicht festgelegt und könnten jederzeit eintreten (z. B. als Ergebnis von **Response.Flush**). Alle anderen Ereignisse sind sequenziell.

Sie müssen kein HTTP-Modul implementieren, nur um eines dieser Ereignisse einbinden zu können. Sie können auch **Global.asax** Ereignishandler hinzufügen. Neben den in Tabelle 1 aufgeführten Ereignissen (die alle von einzelnen HTTP-Modulobjekte eingebunden werden können) löst das **HttpApplication**-Objekt auch die Handler **Application_OnStart** und **Application_OnEnd** aus, die ASP-Entwicklern wohl vertraut sein werden. Diese können nur innerhalb von **Global.asax** verarbeitet werden. Und schließlich können Sie in **Global.asax** auch benutzerdefinierte Ereignishandler für Ereignisse implementieren, die von einzelnen HTTP-Modulobjekten ausgelöst werden. Beispielsweise löst das Sitzungsstatusmodul die Ereignisse **Session_OnStart** und **Session_OnEnd** aus.

Implementieren eines benutzerdefinierten HTTP-Moduls

▶ **So erstellen Sie ein eigenes HTTP-Modul und fügen dieses in die Verarbeitungspipeline von ASP.NET ein**

1. Erstellen Sie eine Klasse, die **IHttpModule** implementiert.
2. Speichern Sie die Assembly, die das Modul enthält, im Unterverzeichnis **\bin** der Anwendung. Alternativ können Sie sie auch im globalen Assemblycache installieren.
3. Fügen Sie wie nachstehend gezeigt das **<HttpModules>**-Element zu **Web.config** der Anwendung hinzu.

```
<system.web>
  <httpModules>
    <add name="modulename"
         type="namespace.classname,assemblyname" />
  </httpModules>
</system.web>
```

Implementieren eines benutzerdefinierten HTTP-Handlers

Möglicherweise müssen Sie einen benutzerdefinierten HTTP-Handler implementieren, beispielsweise, um die Verarbeitung von Dateien mit der Dateinamenerweiterung DATA zu steuern.

▶ **So implementieren Sie einen benutzerdefinierten HTTP-Handler**

1. Fügen Sie der IIS-Metabasis eine Zuordnung hinzu, um die Dateinamenerweiterung DATA der ASP.NET ISAPI-Erweiterung (**Aspnet_isapi.dll**) zuzuordnen.

   Klicken Sie im IIS-MMC-Snap-In mit der rechten Maustaste auf das virtuelle Verzeichnis der Anwendung, klicken Sie auf die Schaltfläche **Konfiguration**, und klicken Sie dann auf **Hinzufügen**, um in **C:\Winnt\Microsoft.Net\Framework\v1.0.3705\aspnet_isapi.dll** eine neue Zuordnung für Dateien vom Typ DATA zu erstellen.

   Hinweis: Wenn Sie beim Hinzufügen der Zuordnung das Kontrollkästchen **Prüfen, ob Datei existiert** aktivieren, muss die Datei physikalisch vorhanden sein. Dies wird der Fall sein, es sei denn, Sie verfügen über virtualisierte Pfade, die nicht auf eine physikalische Datei verweisen. Virtualisierte Pfade, die auf REM oder SOAP enden, werden von .NET Remoting verwendet.

   ---

2. Erstellen Sie eine Klasse, die **IHttpHandler** implementiert (und optional **IHttpAsyncHandler**, sofern Anforderungen asynchron verarbeitet werden sollen).
3. Speichern Sie die Assembly, die den Handler enthält, im Unterverzeichnis **\bin** der Anwendung. Alternativ können Sie sie auch im globalen Assemblycache installieren.
4. Fügen Sie den Handler der Verarbeitungspipeline hinzu, indem Sie der Datei **Web.config** der Anwendung einen **<httpHandlers>**-Abschnitt hinzufügen.

```
wird der Fall sein
<system.web>
  <httpHandlers>
    <add verb="*" path="*.data" type="namespace.classname, assemblyname" />
  </httpHandlers>
</system.web>
```