2

Fundamentos do ASP.NET

ASP.NET é a nova versão da tecnologia Active Server Pages utilizada para desenvolver páginas Web com conteúdo dinâmico. Ao mesmo tempo em que ASP.NET possui sintaxe compatível com o ASP 3.0, também possui um novo modelo de programação orientado a objeto. As páginas ASP.NET são compiladas, e não interpretadas como nas versões anteriores, e podem ser implementadas utilizando-se Visual Basic.NET, C# (lê-se C Sharp) e JScript.NET. Os exemplos deste livro estão em Visual Basic.NET.

Por meio do ASP.NET pode-se utilizar todos os recursos disponíveis no Framework .NET, como o ambiente de runtime CLR, o tipo comum de dados e a biblioteca de classes .NET. Em vez de se limitar aos seis objetos ASP clássicos, o ASP.NET dispõe de uma ampla variedade de componentes úteis que podem ser utilizados ou estendidos. Além disso, é possível utilizar componentes desenvolvidos utilizando a tecnologia COM/COM+, embora aspectos de desempenho devam ser considerados.

O ASP.NET possui dois modelos de programação:

- **Web Forms:** utilizado para construir páginas Web baseadas em formulários. Pode-se utilizar controles de Servidor para a criação de elementos de interface, programá-los e reutilizá-los dentro da aplicação.
- **Web Services:** utilizado para acessar componentes remotamente. Ao usar Web Services, pode-se expor interfaces para acessar dados ou regras de negócio, que podem ser manipulados por clientes ou outras aplicações. Web Services permite a troca de dados em um cenário cliente-cliente ou servidor-servidor, usando protocolos-padrão como HTTP e XML para transportar dados.

Por que uma nova versão ASP?

A seguir, algumas razões que motivaram a implementação de uma nova versão ASP.

Código com estruturação confusa

No ASP é possível a utilização de códigos HTML, Script, componentes e outros elementos misturados na página, dificultando o desenvolvimento, a manutenção e a reutilização de código. O ASP.NET oferece o recurso chamado de "Code Behind", que possibilita colocar a parte de design da página e a parte de código em dois arquivos distintos. O design pode ser feito em paralelo com a programação. Por outro lado, também é possível colocar o código dentro da própria página de design, se o desenvolvedor assim desejar.

Baixa produtividade

No ASP é necessário escrever código para praticamente tudo, como, por exemplo, para manter o estado dos campos de um formulário ou validar a entrada de dados do usuário. O ASP.NET introduz um modelo de componente, com controles baseados no servidor e orientados a eventos, o que torna a programação similar à maneira como é feita no Visual Basic.

Páginas interpretadas

No ASP as páginas são interpretadas, ou seja, cada linha do arquivo é lida, interpretada e transformada em HTML, o qual é enviado para o browser que solicitou a página. No ASP.NET as páginas são compiladas em um executável (DLL), o que pode acarretar um ganho significativo de performance. Além disso, pode-se utilizar qualquer linguagem que suporte o Framework .NET. O Visual Studio.NET oferece o VB.NET, C# e o C++ como opções de instalação. No ASP é possível utilizar somente linguagens de script, como o VbScript e o JScript.

Diversidade de browsers

Com a utilização de dispositivos diferentes para acesso à Internet, como PDAs, aparelhos de TV, console de jogos, celulares, entre outros, surge a necessidade de considerar diferentes saídas para uma mesma aplicação, como, por exemplo utilizando formato HTML, WML, XML e HDML. O ASP.NET oferece um modelo de programação unificado para o desenvolvimento de aplicações para PCs e dispositivos móveis. Utilizando controles especiais e extensões do ASP.NET, pode-se desenvolver aplicações que poderão ser acessadas por meio de diferentes dispositivos. Para o desenvolvimento de aplicações para dispositivos móveis, deve-se utilizar o Mobile Internet Toolkit, que pode ser baixado em http://www.msdn.microsoft.com/vstudio/device/mitdefault.asp.

Além desses pontos, pode-se destacar a necessidade de ferramentas melhores para depuração de código e layout das páginas, recursos limitados de segurança, dificuldades na distribuição e configuração da aplicação, falta de suporte para Web Farm e cache do lado do servidor.

Novos recursos do ASP.NET

O ASP.NET oferece alguns novos recursos como:

- **Controles de Servidor que mantêm o estado:** páginas ASP.NET utilizam controles do lado do servidor para automatizar o gerenciamento do estado na página e reduzir o volume de código.
- **Controles HTML que são executados no servidor:** novos controles HTML que são executados no servidor e produzem código HTML. Pode-se acessar por meio de código as propriedades e os métodos dos controles durante a execução. Os eventos dos controles podem ser detectados pelo browser e o respectivo código pode ser executado no servidor em resposta a esses eventos.
- **Controles de Interface:** controles de interface mais sofisticados executados no servidor, que podem ser utilizados para criar elementos de interface mais complexos na página, sem a necessidade de utilização de controles ActiveX no cliente. O ASP.NET possui controles como Calendar, Grids, tabelas e listas, que podem ser vinculados diretamente a uma fonte de dados.
- **Web Services (Serviços Web):** permitem que componentes sejam disponibilizados como serviços utilizando protocolos-padrão da Internet.
- **Configuração e Distribuição da aplicação simplificados:** a configuração da aplicação pode ser feita por meio de arquivos no formato XML. Os componentes não precisam mais ser registrados no servidor e as aplicações podem, por exemplo, ser distribuídas utilizando comandos de cópia de arquivo ou FTP.
- **Sessões mais escaláveis:** recurso de compartilhamento de informações via sessão (session) mais escalável e com suporte de sessão para Web Farm.
- **Debug e Trace melhorados:** recursos de tratamento de erros, depuração e rastreamento melhorados. A depuração de código no ASP.NET não se limita somente aos comandos Response.write e Response.End.
- **Segurança:** recursos mais flexíveis de segurança para autenticação e autorização do usuário.
- **Cache:** cache do lado do servidor permite que sejam armazenados valores e objetos para utilização nas páginas.
- **ADO x ADO.NET:** substituição do conceito de Recordset do ASP pelo DataSet. O ADO.NET possui uma arquitetura de dados desconectada e integração nativa com XML.
- **Fim do "DLL Hell":** os componentes desenvolvidos na plataforma .NET dispensam o registro no Windows. Uma simples cópia da DLL (assembly) para o local de destino já possibilita sua utilização, até se esta estiver em uso. Além disso, é possível utilizar componentes COM, ou seja, componentes criados pelo Visual Basic 6.0 ou qualquer outra ferramenta de geração de componentes no padrão ActiveX.

Porém, esses componentes serão executados fora do ambiente gerenciado do Framework .NET, ou seja, qualquer erro que ocorrer no componente ficará sob a responsabilidade do próprio sistema operacional.

- **Biblioteca de componentes:** disponibilização de componentes para o desenvolvimento de aplicações Web por meio da Biblioteca de Classes do Framework .NET. Recursos para envio de e-mail, criptografia, contadores, calendário, upload, acesso ao registro de eventos do servidor, MSMQ, acesso a recursos de rede e banco de dados, por exemplo.

Linguagens suportadas no ASP.NET

O ASP.NET pode utilizar qualquer linguagem compatível com o Framework .NET. Inicialmente estão disponíveis as linguagens VB.NET, C# e JScript.

Para definir a linguagem a ser utilizada, devemos utilizar a seguinte sintaxe:

```
<script language="VB" runat="server">
```

Ou em C#:

```
<script language="C#" runat="server">
```

Ferramentas de desenvolvimento

O Framework .NET oferece compiladores de linha de comando, que podem ser utilizados a partir de uma janela DOS, especificando-se os parâmetros corretos.

Todo o código pode ser desenvolvido no NotePad, ou em qualquer outro editor de texto. Porém, pode-se utilizar o Visual Studio.NET, que oferece um ambiente de desenvolvimento visual, reconhecimento da sintaxe ASP.NET, recursos como IntelliSense (que já existia no Visual Studio 6.0, que exibe todos os argumentos e seus tipos quando são digitados o nome de um método e um parêntese de abertura), Clipboard Ring (área de transferência especial), Outlining e Hiding (permite visualizar a estrutura resumida do código), navegação pelo texto (por meio de botões de navegação pode-se navegar por locais já visitados do código), Word wrap (quebra de linha automática) e endentação inteligente.

Outros editores que também podem ser utilizados:

- WebMatrix (www.asp.net) é uma ferramenta gratuita da Microsoft. O livro Desenvolvendo Aplicações ASP.NET com Web Matrix, da Novatec Editora, é uma boa opção para quem quiser aprender a utilizá-la.
- UltraEdit32 (www.ultraedit.com).
- EditPlus (www.editplus.com).
- HomeSite (www.macromedia.com).
- SharpDevelop (www.icsharpcode.net).
- Boxer Text Editor (www.boxersoftware.com).
- Frst Page 2000 (www.pvrsoft.com).

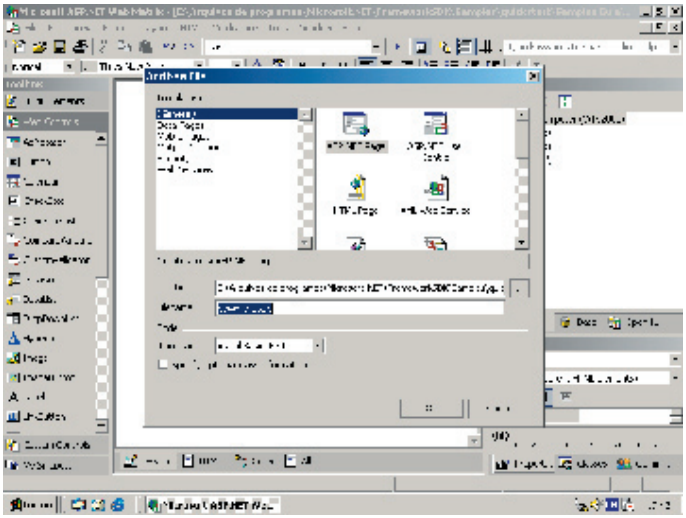

Figura 2.1 – Tela inicial do We Matrix.

Arquitetura do ASP.NET

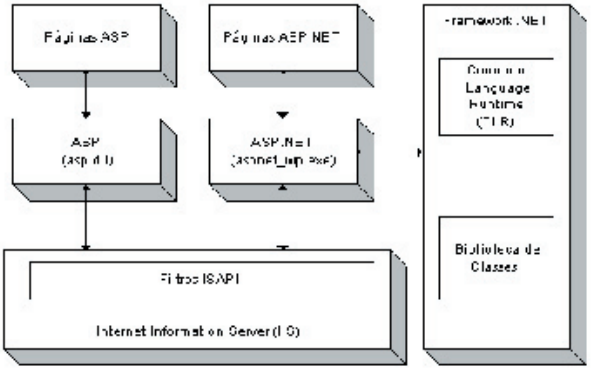

Figura 2.2 – Visão geral da arquitetura do ASP.NET.

Nas versões anteriores do ASP, a página era interpretada sempre que requisitada (com exceção das páginas armazenadas no cache do servidor, que já foram interpretadas). No modelo ASP.NET, o código é compilado para MSIL, transformando a página em um objeto, armazenando-o no cache do servidor e tornando-o disponível para ser executado. A página é compilada quando é criada ou alterada, porém a compilação só ocorre realmente quando a página é solicitada. Mesmo que a página não possua nenhum recurso ASP.NET e tenha apenas HTML comum, ainda assim ela será compilada dentro do modelo .NET.

Quando uma página ASP.NET é requisitada, o servidor Web direciona a requisição para o executável **aspnet_wp.exe**, para que a requisição seja atendida. O resultado é devolvido para o servidor Web, que, por sua vez, o envia para o browser que solicitou a página.

A versão compilada em MSIL fica armazenada em uma DLL dentro de ***Drive:\WINNT\Microsoft.Net\Framework\v.1.x\Temporary ASP.NET files***\. Pode-se visualizar o conteúdo dessa DLL no MSIL Disassembler, digitando ILDASM na linha de comando, como mostrado na figura 2.3.

O ASP.NET utiliza uma extensão de arquivo diferente de suas versões anteriores (.aspx em vez de .asp) e possui um ambiente de execução totalmente separado (o Asp.dll não foi modificado). Isso significa que páginas .asp e .aspx podem coexistir em um mesmo ervidor e em uma mesma aplicação.

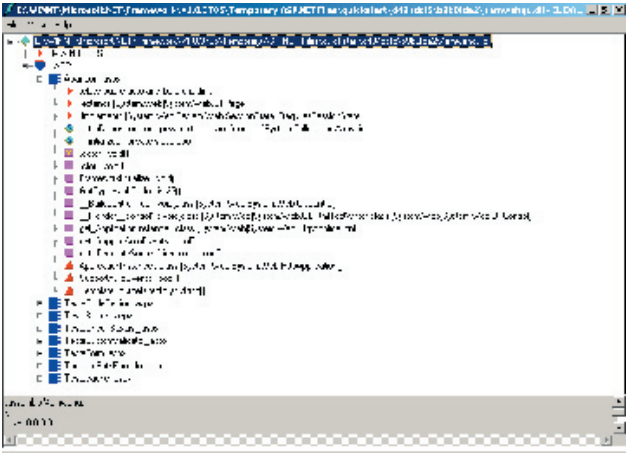

Figura 2.3 – Visualizando informação do assembly com o ILDasm.

Páginas ASP.NET

Uma página ASP.NET é um arquivo texto com a extensã .aspx, como no exemplo a seguir:

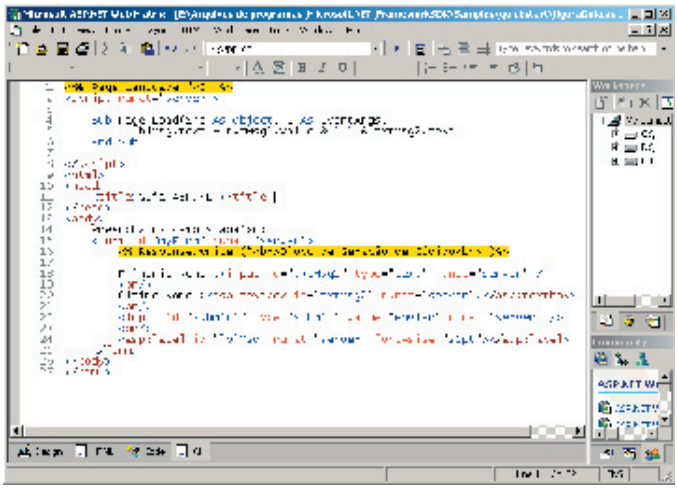

Figura 2.4 – Estrutura de uma página ASP.NET.

Essa página destaca os elementos mais comuns encontrados em uma página ASP.NET:

Linha	Descrição
1	Diretiva <% @ Page %> que especifica parâmetros de compilação da página.
2-8	Bloco de Declaração de Código, onde podem ser colocadas a funcionalidade da página e rotinas de tratamento de eventos.
14	Texto estático.
15	Declaração de um formulário Web, utilizando o controle de servidor "form".
16	Bloco de Geração de Código, onde pode ser inserido código ASP.NET no meio de código HTML.
18-24	Código contendo os controles de servidor (HTML e Web).

Com a introdução de controles executados no servidor (Server Controls), as páginas ASP sofreram uma mudança na maneira de serem programadas. O novo modelo utiliza a programação orientada a eventos, dessa forma, os eventos que acontecem em um controle no cliente podem ser detectados pelo servidor e uma ação pode ser tomada em resposta ao evento. Além disso, no ASP.NET uma página é representada por uma instância da classe Page, que é automaticamente criada toda vez que é feita a requisição da página.

Classe Page

Toda página ASP.NET solicitada é compilada como um objeto da classe Page, e a instância dessa classe é armazenada no cache do servidor, mesmo que a página contenha somente texto HTML. A classe Page funciona como um container para todos os controles que fazem parte da página solicitada, disponibilizando seus métodos, propriedades e eventos, os quais podem ser acessados diretamente em uma página ASP.NET, como, por exemplo: Validate em vez de Page.Validate.

Namespace: System.Web.UI

Assembly: System.Web (System.Web.dll)

Construtor

New Page ()

Cria uma nova instância da classe Page.

Propriedades públicas

A classe Page possui as seguintes propriedades públicas, além das propriedades herdadas da classe Control.

Application

Retorna uma referência a um objeto do tipo Application (HttpApplicationState).

Public ReadOnly Property Application As HttpApplicationState

Cache

Retorna uma referência a um objeto do tipo Cache. Uma instância da classe Cache é criada para cada aplicação, e ela permanece disponível durante todo o ciclo de vida desta. O Cache pode ser utilizado para armazenar valores entre requisições de páginas, os quais serão visíveis para todos os usuários. No ASP.NET, pode-se utilizar o cache do servidor tanto para armazenamento da saída de uma página dinâmica, por meio da diretiva @OutPutCache, como para o armazenamento de valores específicos.

Public ReadOnly Property Cache As Cache

Exemplo:

```
Cache("txt1") = txtName.value
```

ClientTarget

Define ou retorna um valor que permite alterar a detecção automática de browser. Pode ser utilizado para especificar os recursos suportados pelo browser, os quais devem ser considerados para gerar a saída da página (por exemplo, se o browser suporta DHTML). O valor definido nessa propriedade deve corresponder ao valor especificado na seção <clientTarget> do arquivo de configuração Web.Config. Ao definir um valor para essa propriedade, a detecção automática de browser é desabilitada.

Public Property ClientTarget As String

EnableViewState

Define ou retorna um valor indicando se a página (e todos os controles contidos nela) deve manter o seu estado (View State) entre requisições da página (True) ou não (False) (default = True).

Overrides Public Property EnableViewState As Boolean

ErrorPage

Define ou retorna a página que deve ser carregada, caso seja detectado um erro (Exception) não tratado.

Public Property ErrorPage As String

ID

Define ou retorna um identificador da instância da classe Page da página em execução.

Overrides Public Property ID As String

IsPostBack

Indica se a página está sendo recarregada após um "round-trip" (ida e volta ao servidor) ou se está sendo carregada pela primeira vez, permitindo o uso do evento Page_Load para definir valores iniciais para os controles.

Public ReadOnly Property IsPostBack As Boolean

Valor	Descrição
True	Indica que a página foi carregada após um "round-trip".
False	Indica que a página está sendo carregada pela primeira vez ou por meio do botão "Refresh ".

Exemplo:

```
<%@ Page Language="VB" %>
<html>
<head>
   <title>IsPostBack</title>
   <script language="VB" runat="server">
      Sub Page_Load(Src As Object, E As EventArgs)
         if not isPostBack then
            response.write ("Primeira execução da página")
         end if
      End Sub
   </script>
</head>
<body> </body>
</html>
```

IsValid

Indica se a validação efetuada pelos controles de validação ocorreu com sucesso (True). Caso a página não possua controles de validação, retornará True.

Public ReadOnly Property IsValid As Boolean

Request

Retorna uma referência ao objeto Request (HttpRequest).

Public ReadOnly Property Request As HttpRequest

Response

Retorna uma referência ao objeto Response (HttpResponse).

Public ReadOnly Property Response As HttpResponse

Server

Retorna uma referência ao objeto Server (HttpServerUtility).

Public ReadOnly Property Server As HttpServerUtility

Session

Retorna uma referência ao objeto Session (HttpSessionState).

Overridable Public ReadOnly Property Session As HttpSessionState

SmartNavigation

Define ou retorna um valor indicando que o recurso "Smart Navigation" está habilitado (True) ou não (False). Smart Navigation é um recurso suportado pelo browser Internet Explorer 5 (ou superior), que oferece funcionalidades de navegação, tais como: persistência da posição da barra de rolagem entre páginas, persistência do foco em elementos da página durante a navegação e armazenamento do último estado da página no histórico do browser. O recurso Smart Navigation pode ser habilitado pelo atributo SmartNavigation da diretiva @Page.

Public Property SmartNavigation As Boolean

Trace

Retorna uma referência ao objeto TraceContext, que mostra detalhes de execução da página, caso este tenha sido habilitado. A depuração (Tracing) pode ser habilitada por meio da diretiva @Page: <%@ Page trace="true" %>.

Public ReadOnly Property Trace As TraceContext

User

Obtém informações sobre o usuário que está fazendo a requisição da página. Retorna um objeto do tipo IPrincipal (System.Security.Principal) que possui a propriedade Identity.

Public ReadOnly Property User As IPrincipal

Identity expõe as seguintes propriedades:

Propriedade	Descrição
AuthenticationType	Retorna o tipo de autenticação: None, Windows, Forms ou Passport.
IsAuthenticated	Retorna true se o usuário foi autenticado com sucesso, caso contrário, false.
Name	Retorna o nome do usuário autenticado.

Exemplo:

```
response.write (“Usuário = “ & user.identity.name)
response.write (“</br>Tipo de autenticação = “ & user.identity.AuthenticationType)
if (user.identity.isAuthenticated) then
    response.write (“<br>Usuário autenticado.”)
else
    response.write (“</br>Usuário não autenticado.”)
end if
```

Validators

Retorna uma coleção contendo todos os controles de validação da página.

Public ReadOnly Property Validators As ValidatorCollection

Visible

Define ou retorna um valor indicando se o objeto Page será renderizado, ou seja, se a página estará visível (True = default) ou não (False).

Overrides Public Property Visible As Boolean

Exemplo das propriedades da classe Page

```
<%@ Page Language=”VB” %>
<html>
<head>
<title> Livro ASP.NET </title>
<script language=”VB” runat=”server”>
   Sub Page_Load(Src As Object, E As EventArgs)
      response.write (“<br>Propriedades da Classe Page”)
      response.write (“<br> Cache = “ & Convert.ToString(Page.Cache))
      response.write (“<br> ClientTarget = “ & Convert.ToString(Page.ClientTarget))
      response.write (“<br> EnableViewState = “ & Convert.ToString(Page.EnableViewState))
      response.write (“<br> SmartNavigation = “ & Convert.ToString(Page.SmartNavigation))
      response.write (“<br> User = “ & Convert.ToString(Page.User))
   End Sub
</script>
</head>
<body>
</body>
</html>
```

Propriedades protegidas

A classe Page possui a seguinte propriedade protegida, além das propriedades protegidas herdadas da classe Control:

Context

Retorna o objeto HttpContext associado à página.

Overrides Protected ReadOnly Property Context As HttpContext

Métodos públicos

A classe Page possui os seguintes métodos públicos, além dos métodos públicos herdados das classes Control e Object:

DesignerInitialize

Utilizado por ferramentas de desenvolvimento (RAD) para inicializar valores da instância da classe Page.

GetPostBackClientEvent

Retorna uma string que representa a função script cliente que originou o envio (postback) do formulário.

Public Function GetPostBackClientEvent(ByVal *controle* **As Control, _**
ByVal *argumento* **As String) As String**

Parâmetro	Descrição
controle	Controle que recebe o envio (postback) do evento cliente.
argumento	Argumento passado para o método IPostBackEventHandler.RaisePostBackEvent.

GetPostBackClientHyperlink

Retorna o nome da função do lado cliente e o ID do controle do servidor que processou a função.

Public Function GetPostBackClientHyperlink(ByVal *controle* **As Control,**
_ByVal *argumento* **As String) As String**

Parâmetro	Descrição
controle	Controle do servidor que processará o envio (postback).
argumento	Argumento passado para o controle do servidor.

GetPostBackEventReference

(Sobrecarregado)

Retorna uma referência à função do lado cliente que pode ser inserida em um manipulador de evento cliente.

Protótipo 1:

Overloads Public Function GetPostBackEventReference(_
ByVal *controle* **As Control) As String**

Parâmetro	Descrição
controle	Controle do servidor que processará o envio (postback).

Protótipo 2:

Overloads Public Function GetPostBackEventReference(ByVal *controle* **As Control, _**
ByVal *argumento* **As String) As String**

Parâmetro	Descrição
controle	Controle do servidor que processará o envio (postback).
argumento	Argumento passado para o controle do servidor.

GetTypeHashCode

Retorna um código, por meio de uma função "hash", gerado para objetos Page em tempo de execução. Esse código é único para o objeto Page dentro da hierarquia de controles (default = 0).

Overridable Public Function GetTypeHashCode() As Integer

IsClientScriptBlockRegistered

Retorna um valor indicando se o bloco de script cliente está registrado (True) ou não (False) na página. Blocos de script são emitidos no topo da página ASP.NET. Para emitir blocos de script no rodapé da página, deve-se utilizar blocos de script "startup".

Public Function IsClientScriptBlockRegistered(ByVal *chave* **As String) As Boolean**

Parâmetro	Descrição
chave	Chave do script cliente.

IsStartupScriptRegistered

Retorna um valor indicando se o bloco de script "startup" cliente já está registrado (True) ou não (False) em uma página. Utilizado antes da chamada do método Page.RegisterClientScriptBlock para evitar que o script seja registrado novamente. Blocos de script "startup" são emitidos no rodapé da página ASP.NET e devem ser utilizados na inicialização da página. Pelo fato de serem inseridos no rodapé da página, os elementos a que o bloco de script faz referência necessariamente terão sido carregados antes de o script ser executado. Para emitir blocos de script no topo da página, deve-se utilizar Blocos de script.

Public Function IsStartupScriptRegistered(ByVal *chave* **As String) As Boolean**

Parâmetro	Descrição
chave	Chave do script startup cliente.

MapPath

Retorna o caminho físico para um caminho virtual (absoluto ou relativo) correspondente no servidor.

Public Function MapPath(ByVal *virtualPath* **As String) As String**

Parâmetro	Descrição
virtualPath	Especifica o caminho virtual a ser mapeado para um caminho físico. Se o caminho inicia com (/) ou (\), assume ser um caminho virtual absoluto. Caso contrário, assume ser um caminho relativo ao diretório físico no qual o arquivo .aspx está sendo processado.

RegisterArrayDeclaration_

Declara um array utilizando linguagem de script em uma página. Pode ser usado para que o elemento renderizado por um controle de servidor seja inserido em um array no script do lado cliente.

Esse array permite tratar elementos do mesmo tipo de maneira uniforme, facilitando o acesso a estes. Por exemplo, os controles de validação utilizam um array chamado Page_Validators.

Public Sub RegisterArrayDeclaration(ByVal *NomeArray* **As String, _**
ByVal *ValorArray* **As String)**

Parâmetro	Descrição
nomeArray	Nome do array no qual o valor será declarado.
valorArray	Valor a ser colocado no array.

RegisterClientScriptBlock

Insere blocos de script no cliente que podem ser utilizados pelos controles de servidor.

Overridable Public Sub RegisterClientScriptBlock(ByVal *chave* **As String, _**
ByVal *script* **As String)**

Parâmetro	Descrição
chave	Chave única que identifica o bloco de script.
script	Conteúdo do script que será inserido no cliente.

Exemplo:

```
<%@ Page Language="vb" %>
<script runat="server">
    Sub Page_Load( sender as Object,e as EventArgs)
        'Script que será enviado para o cliente
        Dim strScript as String = "<script language=JavaScript> function SubOk() {"
        strScript += "alert('O botão foi clicado !!');}<"
        strScript += "/"
        strScript += "script>"
        If(Not IsClientScriptBlockRegistered("Script1"))
            RegisterClientScriptBlock("Script1", strScript)
        End If
    End Sub
</script>
<html>
<head>
</head>
<body topmargin="20">
    <form id="frmTeste" runat="server">
    <input style="WIDTH: 51px; HEIGHT: 24px" onclick="SubOk();" type="button" size="51"
    value="Ok" />
    </form>
</body>
</html>
```

RegisterHiddenField

Insere um campo oculto (hidden) em um formulário. O campo será enviado para a página quando o controle HtmlForm for renderizado.

Overridable Public Sub RegisterHiddenField(ByVal *nomeCampoOculto* **As String, _**
ByVal *valorCampoOculto* **As String)**

Parâmetro	Descrição
nomeCampoOculto	Nome do campo oculto a ser criado.
valorCampoOculto	O valor a ser associado à propriedade "value" do campo oculto.

Exemplo:

```
<%@ Page Language="VB" %>
<script runat="server">
   Sub Page_Load(sender As Object, e As EventArgs)
      RegisterHiddenField("NomeCampoOculto", "Valor do Campo")
   End sub
</script>
```

RegisterOnSubmitStatement

Usado para acessar o evento cliente OnSubmit em uma página.

Public Sub RegisterOnSubmitStatement(ByVal *chave* **As String, _**
ByVal *script* **As String)**

Parâmetro	Descrição
chave	Chave única que identifica o bloco de script.
script	Bloco de script que será acionado no evento OnSubmit.

Exemplo:

```
<%@ Page Language="VB" %>
<script runat="server">
   Sub Page_Load(sender As Object, e As EventArgs)
      RegisterOnSubmitStatement("submit", "document.write('O evento Submit foi disparado !!')")
   End sub
</script>
<html>
<head>
</head>
<body>
   <form name=frmTeste runat="server">
      <input type="submit" value="Submit" />
   </form>
</body>
</html>
```

RegisterRequiresPostBack

Registra um controle, exigindo o tratamento de envio para o servidor (post-back).

Public Sub RegisterRequiresPostBack(ByVal *controle* **As Control)**

RegisterRequiresRaiseEvent

Registra um controle, exigindo que um evento seja disparado quando o controle for processado na página.

Overridable Public Sub RegisterRequiresRaiseEvent(_
ByVal *controle* **As IPostBackEventHandler)**

RegisterStartupScript

Registra blocos de script "startup" em uma página, evitando que blocos duplicados de código de script sejam enviados ao cliente.

Overridable Public Sub RegisterStartupScript(ByVal *chave* **As String, _**
ByVal *script* **As String)**

Parâmetro	Descrição
chave	Chave única que identifica o bloco de script.
script	Conteúdo do bloco de script.

RegisterViewStateHandler

Utilizado para persistir o "View State" de uma página. Normalmente somente o objeto HtmlForm chama esse método.

Public Sub RegisterViewStateHandler()

Validate

Faz com que todos os controles de validação contidos em uma página executem suas validações. Esse método é executado pelos controles que possuem a propriedade CausesValidation com valor True.

Overridable Public Sub Validate()

VerifyRenderingInServerForm

Utilizado para confirmar se um controle que requer um formulário (HtmlForm) foi devidamente inserido nele. Se o controle não estiver entre as tags de formulário, uma exceção será gerada.

Overridable Public Sub VerifyRenderingInServerForm(ByVal *controle* **As Control)**

Métodos protegidos

A classe Page possui os seguintes métodos protegidos, além dos métodos protegidos herdados das classes Control, TemplateControl e Object:

CreateHtmlTextWriter

Cria um objeto *obj* para renderizar o conteúdo da página.

Overridable Protected Function CreateHtmlTextWriter(ByVal *obj* **As TextWriter) As HtmlTextWriter**

DeterminePostBackMode

Determina o tipo de solicitação feito para a página.

Overridable Protected Function DeterminePostBackMode() As NameValueCollection

LoadPageStateFromPersistenceMedium

Carrega as informações de estado para o objeto Page. Esse método deve ser sobreescrito para que o estado possa ser carregado com outra fonte que não seja um campo oculto.

Overridable Protected Function LoadPageStateFromPersistenceMedium() As Object

RaisePostBackEvent

Notifica o controle que causou o envio de que ele deve atender a um evento.

Overridable Protected Sub RaisePostBackEvent(ByVal sourceControl As IPostBack-EventHandler, ByVal eventArgument As String)

SavePageStateToPersistenceMedium

Salva qualquer informação de "View State" da página por meio do objeto especificado em *viewState*.

Overridable Protected Sub SavePageStateToPersistenceMedium(ByVal *viewState* As Object)

Eventos

A classe Page possui os seguintes eventos públicos herdados das classes TemplateControl e Control, listados pela ordem de execução:

Init

Ocorre quando a página é inicializada. É o primeiro passo no ciclo de vida da página e qualquer informação de inicialização pode ser colocada nesse evento. Informação de "View State" não está disponível nesse evento (pois ainda não foi gerada), assim como o acesso a outros controles da página não deve ser feito, pois estes ainda não estão disponíveis durante esse evento (herdado da classe Control).

Load

Ocorre toda vez que a página é carregada. Utilize a propriedade IsPostBack para verificar se é a primeira vez que a página é carregada (herdado da classe Control).

DataBinding

Ocorre quando o método DataBind é chamado na página. O método DataBind vincula uma fonte de dados ao controle (herdado da classe Control).

PreRender

Ocorre antes que qualquer saída seja enviada para o browser. Mudanças no estado do controle (View State) podem ser atualizadas durante esse evento (herdado da classe Control).

Unload

Ocorre toda vez que o processamento da página é finalizado. Qualquer ação final como fechar arquivos, fechar conexões com banco de dados ou descartar objetos pode ser feita durante esse evento (herdado da classe Control).

Dispose

Ocorre quando o objeto Page é liberado da memória (herdado da classe Control).

Demais eventos

Além dos eventos listados anteriormente, os eventos a seguir (não determinísticos) podem ocorrer em uma página ASP.NET:

AbortTransaction

Ocorre quando é cancelada uma transação (herdado da classe TemplateControl).

CommitTransaction

Ocorre quando uma transação é concluída com sucesso (herdado da classe TemplateControl).

Error

Ocorre quando um erro (exception) não tratado é detectado (herdado da classe TemplateControl).

Sintaxe das páginas ASP.NET

Além de conteúdo estático, as páginas ASP.NET suportam oito diferentes tags com suas respectivas sintaxes:

Sintaxe de geração de código: <% %> e <%= %>

Utilizada para delimitar **blocos de geração de código** (rendering code). A tag <%= "Texto" %> equivale à <% Response.write ("Texto") %> em VB.NET.

Exemplo:

```
<%="Bloco de geração de código"%>
```

Sintaxe de declaração de código: <script runat="server">

Utilizada para delimitar **blocos de declaração de código** que contêm elementos que serão membros (métodos ou propriedades) da classe Page quando a página for compilada. Todas as funções e variáveis globais da página devem ser declaradas dentro dessa tag.

Exemplo:

```
<%@ Page Language="vb" %>
<script runat="server">
    ' Bloco de declaração de código
    Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
        lblMsg.Text = "Oi " & Nome.Text & ". Você mora na " & Endereco.Text & "."
    End Sub
</script>
<html>
<head>
```

```
</head>
<body>
    <form runat="server">
        <table border="0">
            <tr>
            <td>Nome</td>
            <td><asp:TextBox id="Nome" runat="server"></asp:TextBox></td>
            </tr>
            <tr>
            <td>Endereço</td>
            <td><asp:TextBox id="Endereco" runat="server" Columns="30" Rows="5"
    TextMode="MultiLine"></asp:TextBox></td></tr>
            <tr>
            <td></td>
            <td><asp:Button id="btnSubmit" onclick="SubmitBtn_Click" runat="server"
    text="Submit"></asp:Button></td></tr>
        </table>
        <p><asp:Label id="lblMsg" runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

Sintaxe de controle de servidor Web

Utilizada para declarar Controles de Servidor Web.

Exemplo:

```
<asp:label id="lblMensagem" runat="server"/>
```

Sintaxe de controle HTML

Utilizada para declarar controles HTML que serão processados no servidor. Distingue-se de uma tag HTML-padrão por meio do atributo runat="server".

Exemplo:

```
<span id="spnMensagem" runat="server"/>
```

Sintaxe de Data Binding: <%# %>

O código declarado dentro da tag <%# %> é executado somente quando o método DataBind do container é chamado.

Exemplo:

```
<asp:TextBox id="txtNome" runat="server"
    Nome='<%# DataView1(0)("des_nome_usuario") %>'>
</asp:TextBox>
```

Sintaxe de objeto: <object runat="server" />

Utilizado para declarar e criar instâncias de objetos.

Exemplo:

```
<object id="items" class="System.Collections.ArrayList" runat="server"/>
```

Sintaxe de comentário: <%— comentário—%>

Utilizada para comentar ou desabilitar o código dentro dos blocos de geração de código da página (tags <% %>). O conteúdo dessa tag não é processado. Para comentar o código dentro dos blocos de declaração de código (<script runat=server>) deve-se utilizar o estilo de comentário da linguagem de programação que estiver sendo empregada. Por exemplo, para o VB.NET, deve-se utilizar um caractere de aspas simples para comentar uma linha de cada vez.

Exemplo:

```
<%--
Dim i as integer
for i=0 to 10
    response.write ("Comentário : esse código não será executado !")
next
--%>
```

Sintaxe de Server-Side Include: <!— #Include File="LibData.inc" —>

Permite a inclusão de conteúdo contido em outro arquivo, em qualquer lugar na página ASP.NET. Por default, essa tag pode ser utilizada somente a partir de arquivos com as extensões .shtml, .shtm, .stm, .asp, .asa, .asax e .aspx. Pode-se alterar essas extensões por meio do Internet Services Manager.

Um arquivo pode ser incluído por meio de seu caminho físico ou virtual.

<!— #include *tipoCaminho* = *nomeArquivo***—>**

Parâmetro	Descrição
tipoCaminho	O tipo do caminho do arquivo. Pode ser File ou Virtual.
file	Indica que o parâmetro *nomeArquivo* é um caminho relativo a partir do diretório contendo o arquivo com a diretiva include. O arquivo include pode estar no mesmo diretório ou em um subdiretório, porém não pode estar em um diretório acima do arquivo que contém a diretiva include.
virtual	Indica que o parâmetro *nomeArquivo* é um caminho virtual a partir do diretório virtual da aplicação.
nomeArquivo	O nome do arquivo cujo conteúdo será incluído na página corrente. Deve conter a extensão do arquivo e deve estar definido entre aspas (“).

Exemplo:

```
<!— #include file = “Cabecalho.inc” —>
```

Principais diretivas de página

As diretivas de página definem parâmetros opcionais utilizados pelo ASP.NET para o processamento das páginas e User Controls. Podem ser inseridas em qualquer lugar na página com a seguinte sintaxe:

<%@ diretiva *atributo*=*valor* [*atributo*=*valor*...]**%>**

As seguintes diretivas podem ser utilizadas:

@ Page

Define parâmetros específicos de compilação e processamento para a página. Principais atributos:

Atributo	Descrição
AspCompat	Se **True**, permite o acesso a componentes COM com modelo de thread "single-threaded apartment" (STA) (VB 6.0) (default = false).
Buffer	Habilita (true) ou desabilita (false) o buffer da página (default = true).
ClientTarget	Indica para que tipo de browser os elementos da página devem ser renderizados.
Debug	Utilizada para colocar a página em modo de debug (default = false).
EnableViewState	Habilita (true) ou desabilita (false) o controle de estado dos elementos da página (default = true).
ErrorPage	Especifica a página de erro.
Explicit	Obriga (true) ou não (false) a declaração de variáveis (default = false).
Inherits	Indica a classe utilizada pela página**.**
Language	Especifica a linguagem utilizada na página nas tags <%%>.
Src	Especifica o arquivo contendo o código-fonte.
Trace	Habilita (true) ou desabilita (false) o rastreamento para a página (default = false).

@ Control

Define parâmetros específicos de compilação e processamento para o User Control (.ascx).

<%@ Control *atributo*=*valor* [*atributo*=*valor*...] **%>**

Da mesma forma que a diretiva @ Page, possui uma série de atributos:

Atributo	Descrição
AutoEventWireup	Indica se os eventos da página estão ligados automaticamente (default=True).
ClassName	Especifica o nome da classe da página.
CompilerOptions	String contendo as opções de compilação para a página.
Debug	Indica se a página deve ser compilada com símbolos de debug.
Description	Descrição da página.
EnableViewState	Indica se o "View State" do User Control será mantido entre solicitações (True = default).
Explicit	Determina se a página será compilada utilizando o comando "Option Explicit" do Visual Basic.
Inherits	Define a classe de code-behind do User Control.
Language	Especifica a linguagem utilizada na página.

Atributo	Descrição (cont.)
Strict	Determina se a página será compilada utilizando o comando "Option Strict" do Visual Basic.
Src	Especifica o nome do arquivo-fonte da classe code-behind do User Control.
WarningLevel	Indica o nível de advertência do compilador (0 a 4).

@ Import

Importa um namespace para uma página. Caso a diretiva @Import seja utilizada no arquivo Global.asax, o namespace será importado para toda a aplicação, e suas classes estarão disponíveis para todas as páginas. O namespace importado pode ser parte do Framework .NET ou um namespace definido pelo usuário. Não é preciso importar um namespace para utilizar seus objetos. A importação apenas facilita o acesso ao objeto, porém se pode referenciar o objeto utilizando seu namespace completo, como, por exemplo: Dim obj as System.IO.File.

<%@ Import namespace="*nomeNamespace***" %>**

Exemplo:

```
<%@ Import Namespace="System.Net" %>
```

Os seguintes namespaces são importados automaticamente para todas as páginas ASP.NET:

- 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

@ Implements

Indica que a página ou User Control implementará uma interface específica.

<%@ Implements interface="*NomeInterface***" %>**

@ Assembly

Vincula um assembly à página corrente durante a compilação, tornando todas as classes do assembly disponíveis para serem utilizadas na página. Caso a diretiva seja utilizada em um arquivo Global.asax, as classes estarão disponíveis para toda a aplicação.

A referência a um assembly é feita em tempo de compilação (early binding). A partir do momento que a página referenciando o assembly for compilada, o assembly é carregado e fica disponível para toda a aplicação, permitindo vínculo tardio (late binding).

<%@ Assembly Name="*NomeAssembly***" %>**

<%@ Assembly Src="*caminho***" %>**

Atributo	Descrição
Name	Uma string que representa o nome do assembly a ser vinculado na página. A extensão do arquivo não deve ser incluída.
Src	O caminho do arquivo do assembly a ser vinculado.

Não é possível utilizar os atributos Name e Src na mesma diretiva @Assembly. Para utilizá-los, deve-se utilizar uma diretiva para cada atributo.

Os assemblies armazenados no diretório /Bin, abaixo do diretório raiz da aplicação, são automaticamente importados para a aplicação. Para impedir que isso aconteça, deve-se retirar a seguinte linha do arquivo Web.config:

<add assembly="*" />

O arquivo Web.config também pode ser utilizado para importar um assembly para uma aplicação.

Exemplo:

```
<%@ Assembly Name="NomeAssembly" %>
<%@ Assembly Src="ArquivoFonte.vb" %>
```

@ OutputCache

Define a política de cache para a página ou User Control, permitindo o armazenamento da saída da página ou User Control no cache do servidor, ou seja, a saída é gravada em disco rígido na localização especificada em Location. O cache de saída pode ser utilizado para se obter ganhos de performance em páginas muito acessadas.

<%@ OutputCache Duration="*Tempo***" [Location="***Localização***"] %>**

Atributo	Descrição
Duration	Tempo (em segundos) em que a página é armazenada no cache.
Location	Especifica o local de armazenamento da saída (Any, Client, Downstream, None, Server).

Code Behind

O conceito de Code behind ("Por trás do Código") é utilizado para separar as partes de apresentação e lógica do código de uma página ASP.NET, utilizando herança e permitindo a reutilização de código.

Para utilizar o Code Behind é preciso criar uma classe que derive da classe Page (System.Web.UI.Page) ou da classe UserControl (System.Web.UI.UserControl), se for um User Control. Pode-se pôr todo o código que seria colocado dentro do arquivo .aspx dentro da classe do arquivo do Code Behind e deve-se declarar os controles que serão utilizados na página .aspx como públicos no arquivo de Code Behind. No exemplo a seguir, o controle TextBox chamado "Nome" no arquivo .aspx é declarado como público no arquivo do Code Behind. Isso garante que seja possível tratar as propriedades e os eventos dos controles. Em seguida, deve-se adicionar os atributos Inherits e Src ao arquivo .apx, especificando o nome da classe declarada no Code Behind e sua localização, respectivamente.

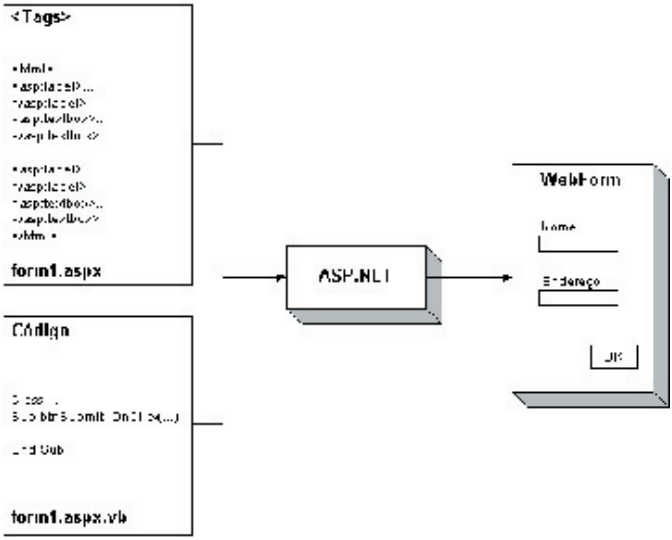

Figura 2.5 – Code Behind: separação da apresentação e lógica de uma página em arquivos distintos.

Exemplo de Code Behind:

Primeiro deve ser criado o arquivo com o "Code Behind" (MeuFonteCodeBehind.vb):

```
Option Strict Off
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Web.UI.HtmlControls

Public Class MeuCodeBehind: Inherits Page
    Public Nome As TextBox
    Public spnSpan1 As HtmlGenericControl
    Public Sub SubmitBtn_Click(Sender As Object, E As EventArgs)
        spnSpan1.InnerHtml = "Olá, " & Nome.Text & "."
    End Sub
End Class
```

E em seguida, pode-se utilizar o código em uma página ASP.NET, como a seguir:

```
<%@ Page Inherits="MeuCodeBehind" Src="MeuFonteCodeBehind.vb" %>
<html>
<body>
    <form runat="server">
    <table>
        <tr>
            <td> Nome: </td>
            <td> <asp:textbox id="Nome" runat="server"/> </td>
        </tr>
        <tr>
            <td></td>
            <td><asp:button id="btnOk" text="Clique aqui" OnClick="SubmitBtn_Click"
    runat="server"/></td>
        </tr>
        <tr>
            <td></td>
            <td><span id="spnSpan1" runat="server" /></td>
        </tr>
    </table>
</form>
</body>
</html>
```

Aplicação ASP.NET

Uma aplicação ASP.NET é o conjunto de todos os arquivos, páginas, handlers, módulos e códigos executáveis que podem ser acessados por meio de um diretório virtual e seus subdiretórios em um servidor Web. Aplicações ASP.NET são entidades totalmente separadas, ou seja, cada aplicação é tratada dentro de um domínio de aplicação do CLR (runtime). Isso significa que cada aplicação pode manter suas próprias características de processamento e configurações, por meio dos arquivos Global.asax e Web.Config.

Cache de Assembly

Uma aplicação ASP.NET utiliza o diretório \bin chamado de "Cache de Assembly", para armazenar os arquivos compilados (DLLs) utilizados. Cada aplicação tem seu próprio diretório, ou pode herdar um de seu diretório-pai. Os assemblies colocados nesse diretório ficam automaticamente disponíveis para a aplicação.

O arquivo Global.asax

É um arquivo opcional com a finalidade de armazenar informações ou objetos que serão utilizados globalmente pela aplicação. O Global.asax deve residir no diretório raiz da aplicação e possui sintaxe e utilidade similar ao arquivo Global.asp do ASP 3.0. O Global.asax é um arquivo em formato-texto que obedece às mesmas regras de compilação e execução de um arquivo .aspx. Quando compilado, é transformado em IL como classe derivada de HttpAplication.

Os arquivos Global.asa e Global.asax podem existir em uma mesma aplicação, e as informações contidas no arquivo Global.asa somente serão visíveis para páginas com extensão .asp e vice-versa.

Por segurança, o arquivo Global.asax não pode ser executado a partir do browser.

Partes do arquivo Global.asax

Um arquivo Globa.asax pode ser composto dos seguintes elementos:

- **Diretivas de aplicação:** especifica parâmetros opcionais de compilação e execução das páginas ASP.NET.
- **Blocos de código:** define os métodos e propriedades que farão parte da classe HttpApplication, que é gerada como resultado da compilação do arquivo Global.asax.
- **Declarações <Object>:** declaração e criação de objetos.
- **Diretivas Server-Side:** inclui o conteúdo de um arquivo especificado no arquivo Global.asax.

Diretivas de aplicação

Três tipos de diretivas podem ser utilizadas: Application, Import e Assembly.

@Application

Define atributos específicos da aplicação utilizados pelo compilador.

Sintaxe:

<%@ Application *atributo*=**"***valor***"** [*atributo*=**"***valor***" ...]%>**

Atributo	Descrição
Description	Descrição em formato-texto da aplicação. Esse valor é ignorado pelo parser e compilador do ASP.NET.
Inherits	Nome da classe a ser derivada. Por padrão, o arquivo Global.asax é compilado como uma classe derivada de HttpApplication, porém é possível especificar, por meio desse atributo, uma outra classe, desde que ela também seja derivada da classe HttpApplication.

Exemplo:

```
<%@ Application Inherits="ObjApp.ClassApp" Description="Descrição da Aplicação" %>
```

@Import

Ver descrição dessa diretiva na página 38.

@Assembly

Ver descrição dessa diretiva na página 39.

Blocos de código

Blocos de código permitem a definição de variáveis, métodos e eventos que farão parte da classe HttpApplication, resultado da compilação do arquivo Global.asax.

Sintaxe:

<script runat="server" [language="*linguagem***"] [src="***nomeArquivoFonte***"]>**

Código ...

</script>

Atributo	Descrição
runat	Especifica que o código será executado no servidor.
language	A linguagem utilizada no bloco de código corrente. Se não for especificada, a linguagem default será utilizada (definida no arquivo de configuração da aplicação).
src	O nome do arquivo contendo o código. Quando utilizado, qualquer código dentro do bloco é ignorado.

Declarações <Object>

A tag <Object> pode ser utilizada para declarar e criar objetos de aplicação e sessão. A tag pode ser utilizada de três diferentes formas, dependendo do tipo do objeto a ser criado:

<object id="*identificador***" runat="server" scope="***escopo***" class="***NomeClasse***">**

<object id="*identificador***" runat="server" scope="***escopo***" progid="***ProgID do Componente***"/>**

<object id="*identificador***" runat="server" scope="***escopo***" classid="***ClassID do Componente***"/>**

Atributo	Descrição
id	Identificador único dentro da página para o objeto a ser criado e usado para referenciar o objeto no código.
runat	Especifica que o código será executado no servidor.
scope	Escopo do objeto a ser criado (default=**pipeline**).
pipeline	Disponível somente dentro da solicitação (request).
application	Disponível para toda a aplicação e o objeto é armazenado dentro da coleção HttpApplicationState.
session	Disponível dentro da sessão e o objeto é armazenado dentro da coleção HttpSessionState.
class	Classe da qual será criada uma instância.
progID	Identificador programático do componente (COM) a ser instanciado.
classID	CLSID do componente (COM) a ser instanciado.

Os atributos class, progID e classId são mutuamente exclusivos e geram um erro quando declarados dentro de uma mesma tag <object>.

Exemplo:

```
<object id="objItens" class="System.Collections.ArrayList" runat="server"/>
```

Eventos do Global.asax

O arquivo Global.asax suporta os eventos expostos pela classe HttpApplication, além dos eventos já suportados nas versões anteriores. Os eventos podem ser declarados da seguinte maneira:

Application_EventName (*Argumentos do Evento*)

A seguir descreveremos os eventos disponíveis que podemos utilizar no global.asax:

Application_Start

Ocorre na primeria vez em que o usuário solicita uma página da aplicação.

Application_End

Ocorre quando o serviço Web é parado no servidor.

Session_Start

Ocorre quando um novo usuário solicita uma página da aplicação, ou seja, quando uma sessão de usuário é iniciada.

Session_End

Ocorre quando a sessão de usuário é encerrada por timeout de sessão (default=20 minutos) ou pela execução do método Abandon (Session.Abandon).

Application_AcquireRequestState

Ocorre quando o ASP.NET obtém o estado corrente associado à solicitação corrente.

Application_AuthenticateRequest

Ocorre quando a identidade do usuário é estabelecida. Pode ser usado para executar qualquer código antes que o usuário seja autenticado.

Application_AuthorizeRequest

Ocorre quando a autorização do usuário é verificada. Pode ser utilizado para executar qualquer código antes de o usuário ter autorização para um recurso, como, por exemplo, autorização para acesso a uma URL.

Application_BeginRequest

Ocorre toda vez que uma solicitação é feita ao servidor Web.

Application_Disposed

Ocorre quando a cadeia de execução de uma solicitação foi completada.

Application_EndRequest

É o último evento a ser executado no atendimento a uma solicitação.

Application_Error

Ocorre quando um erro não tratado é encontrado.

Application_PostRequestHandlerExecute

Ocorre quando um handler ASP.NET (página ou Web Service) termina seu processamento. É o primeiro evento disponível após o handler concluir seu trabalho.

Application_PreRequestHandlerExecute

Ocorre antes do início de execução de um handler ASP.NET.

Application_PreSendRequestContent

Ocorre antes do envio de conteúdo para o cliente.

Application_PreSendRequestHeaders

Ocorre antes do envio de cabeçalhos HTTP para o cliente.

Application_ReleaseRequestState

Ocorre depois que o ASP.NET termina a execução de todos os handlers.

Application_ResolveRequestCache

Ocorre quando o ASP.NET completa um evento de autorização deixando os módulos de cache atenderem à solicitação, em vez da página ou Web Service.

Application_UpdateRequestCache

Ocorre quando o ASP.NET termina a execução de um handler com a finalidade de deixar que módulos de cache armazenem respostas que serão utilizadas para solicitações subseqüentes a partir do cache.

Exemplo:

O seguinte exemplo ilustra como utilizar o evento Error para gravar informação de erro no Log de eventos do NT:

```
<%@ import Namespace="System.Diagnostics" %>
Dim NomeLog As String = "LogTesteAplicacao"
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Cria evento
    If (Not EventLog.SourceExists(NomeLog)) Then
        EventLog.CreateEventSource(NomeLog, NomeLog)
    End If
End Sub

Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    Dim strMensagem As String = "Url " & Request.Path & " Erro: " & Me.Error.ToString()
    ' Cria entrada no log de eventos
    Dim Log As New EventLog()
    Log.Source = NomeLog
    Log.WriteEntry(strMensagem, EventLogEntryType.Error)
End Sub
</script>
```

Diretivas Server-Side Include

Como em uma página ASP.NET comum, a inclusão de conteúdo contido em outro arquivo é permitida no arquivo Global.asax e é realizada por meio da tag Include. Para mais detalhes sobre a sintaxe veja a página 36.

Arquivo Web.Config

O arquivo Web.Config é um arquivo-texto em formato XML utilizado para armazenar informações de configuração do servidor Web, tais como habilitar o estado de sessão (session state), session timeout, buffering, linguagens default, ou mesmo armazenar outras informações de configuração personalizadas, cujos pares de chaves/valores podem ser criados e inseridos no arquivo. No ASP 3.0 essas configurações eram feitas por meio do Internet Information Server (IIS), o que dificultava o acesso programático a estas. No ASP.NET é possível copiar a configuração do servidor Web com seu código e conteúdo, como também alterar essas configurações via FTP.

Da mesma maneira que o arquivo Global.asax, o Web.Config não é um arquivo exigido para que a aplicação funcione. Caso ele não seja encontrado na hierarquia de diretórios da aplicação, será utilizado um arquivo chamado Machine.Config (WINNT\Microsoft.NET\Framework*versão*\CONFIG\Machine.config), com configurações predefinidas que servem para todo o servidor Web. Pode-se criar vários arquivos de configuração (todos chamados Web.config), os quais podem ser colocados em diferentes diretórios da aplicação, e suas configurações terão efeito sobre o diretório corrente e seus respectivos subdiretórios. Arquivos de configurações em diretórios-filho herdam configurações definidas em diretórios-pai, e também podem alterar ou sobrepor essas configurações. Esse sistema hierárquico de herança de configurações é baseado no caminho do diretório virtual, e não no caminho físico.

Da mesma maneira que o arquivo Global.asax, por segurança, o arquivo Web.Config não pode ser acessado via browser.

Estrutura do arquivo Web.Config

O arquivo Web.Config segue regras de sintaxe XML. Toda informação de configuração deve ficar entre as tags <configuration> e </configuration>, que são agrupadas em duas áreas principais: seção de declaração do handler <configSections> e seção de definição de configuração do handler <handlerName>. Um handler é uma classe .NET que implementa a interface IConfigurationSectionHandler.

```
<configuration>
   <configSections>
      <section name="handler_1"/>
      <section name="handler_2"/>
   </configSections>
   < handler_1>
      configuração do handler_1
   </handler_1>
   < handler_2>
      configuração do handler_2
   </handler_2>
</configuration>
```

Configurações-padrão do ASP.NET

O ASP.NET possui os seguintes handlers-padrão que podem ser utilizados para definir as configurações no arquivo Web.Config:

Seção	Descrição
<httpModules>	Configura módulos de HTTP dentro de uma aplicação, os quais participam no processamento de cada solicitação. Utilizações comuns incluem segurança e conexão.
<httpHandlers>	Mapeia URLs de entrada em classes IHttpHandler. Os subdiretórios não herdam essas configurações. Responsável também pelo mapeamento de URLs de entrada em classes IHttpHandlerFactory. Os dados representados nas seções <HttpHandlerFactories> são hierarquicamente herdados por subdiretórios.
<sessionState>	Configura o estado de sessão.
<globalization>	Define as configurações de globalização de uma aplicação.
<compilation>	Define as configurações de compilação de uma aplicação.
<trace>	Configura o rastreamento de uma aplicação.
<processModel>	Configura o modelo de processamento do ASP.NET no servidor Web.
<browserCaps>	Define as configurações do componente de capacidades do browser.

Exemplo de arquivo Web.Config:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <appSettings>
        <add key="dsn" value="localhost;uid=NomeUsuario;pwd=;" />
        <add key="msmqserver" value="server\srvQueue" />
    </appSettings>
    <system.web>
        <!— Compilation —>
        <compilation defaultLanguage="vb" debug="true" />
        <!— Custom Errors —>
        <customErrors mode="On" defaultRedirect="PaginaErroCustomizada.htm" />
        <!— Tracing —>
        <trace enabled="true" requestLimit="10" pageOutput="false" traceMode="SortByTime"
    localOnly="true" />
        <!— Session State —>
        <sessionState mode="StateServer" cookieless="false" timeout="20" />
    </system.web>
</configuration>
```

Acessando informações de configuração nas páginas

Pode-se acessar as configurações definidas no arquivo Web.Config, a partir de uma página, por meio dos objetos intrínsecos ASP.NET. A seguinte linha de código dá acesso ao atributo cookieless da seção <sessionState>:

```
Dim nocookies As Boolean = Session.IsCookieless
```

Ou por meio do método GetConfig da coleção ConfigurationSettings:

```
Dim config As NameValueCollection =  ConfigurationSettings.GetConfig("mysection")
Response.Write("O valor da chave um é " & config("key_one") & "<br>")
```

Configurações de aplicação armazenadas na seção <appSettings> podem ser acessadas da seguinte forma:

```
Dim dsn As String = ConfigurationSettings.AppSettings("dsn")
```

Gerenciamento de estado da aplicação

Páginas ASP.NET são baseadas no protocolo HTTP, e dessa forma não persiste o estado entre solicitações (stateless). Para resolver esse problema, o ASP.NET oferece as seguintes alternativas:

Gerenciamento de estado no lado Cliente

Nenhuma informação de estado é mantida no servidor entre as solicitações. A informação fica armazenada na página ou no computador-cliente.

Cookies

Arquivo-texto armazenado no computador do usuário que pode conter até 4.096 bytes. O cookie é um dos métodos utilizados pelo ASP.NET para identificar a sessão do usuário. Para trabalhar com cookies, é necessário utilizar a classe HttpCookie para armazená-los, e as classes HttpResponse e HttpRequest para enviá-los e recebê-los, respectivamente.

Exemplo:

```
'Lê o cookie armazenado no cliente
If Request.Cookies("UserName") = Null Then
    Dim cookie As New HttpCookie("NomeUsuario")
    cookie.Values.Add("NomeUsuario", "Isabela")
    ...
    'Envia o cookie para o cliente
    Response.AppendCookie(cookie)
End If
```

Campos ocultos (Hidden)

Campos ocultos não ficam visíveis na página e podem ser utilizados para armazenar informações específicas da página. O conteúdo de um campo oculto é enviado na coleção Form. O ASP.NET possui o controle HtmlInputHidden que oferece funcionalidade para campos ocultos.

View State

Recurso utilizado pelo ASP.NET para descrever o estado de um controle de servidor em um dado momento. É utilizado para persistir os valores dos controles, por exemplo, conteúdos de caixa de texto e seleção de botões de rádio entre o envio e retorno de uma página (postback). O ASP.NET automaticamente monitora o View State. Isso significa que se você preencher um formulário HTML e clicar em "submit", os valores ainda estarão na página quando esta voltar do processamento no servidor.

O ASP.NET implementa o View State incorporando campos HTML ocultos na página sempre que um controle de servidor for utilizado (runat=server). Além disso, pode-se armazenar valores no View State por meio do objeto StateBag, da seguinte maneira:

ViewState("*Nome da variável***") =** *valor*

O StateBag é descartado assim que o usuário deixa a página. Os valores são mantidos desde que o Post seja feito para a mesma página.

Query Strings

Possibilita o envio de pequenos volumes de dados de uma página para outra. Para mais detalhes, consultar a propriedade HttpRequest.QueryString, na página 75.

Gerenciamento de estado no lado Servidor

O gerenciamento de estado no servidor pode ser feito por meio dos objetos intrínsecos Application (HttpApplicationState) e Session (HttpSessionState). Além disso, pode-se fazer o gerenciamento de estado da aplicação utilizando o banco de dados. O armazenamento de estado em um banco de dados é especialmente útil para a manutenção do estado em um período de tempo maior ou quando o estado deve ser persistido mesmo que o servidor Web tenha que ser reinicializado. O armazenamento em banco é normalmente utilizado em conjunto com cookies.

Depurando páginas ASP.NET

O suporte para depuração e tratamento de erros, chamados de exceções, foi significativamente melhorado no ASP.NET. Pode-se utilizar ferramentas de depuração e rastreamento, como o CLR Debugger e o Trace. O rastreamento é a coleta de informações sobre a página baseado nas solicitações feitas para esta. Já a depuração refere-se à execução passo-a-passo do código, com a possibilidade da definição de "breakpoints" e visualização do conteúdo de variáveis.

Além disso, pode-se definir páginas de erro personalizadas, como também utilizar a estrutura de tratamento de exceções Try e Catch.

Try e Catch

As instruções Try e Catch são utilizadas para capturar erros e tratá-los dentro da página. Pode-se também utilizar a palavra-chave Throw para gerar erros personalizados, da mesma maneira que o comando err.raise era utilizado no Visual Basic 6.0.

Sintaxe:

```
Try
    [ bloco de código onde poderá ocorrer um erro ]
[ Catch [ exceção [ As type ] ] [ When expressão ]
    [ bloco de tratamento do erro] ]
[ Exit Try ]
...
[ Finally
    [ bloco a ser executado incondicionalmente] ]
End Try
```

Exemplo:

```
' cria o objeto OleDbCommand aqui
Try objCmd.Connection.Open
Catch objEx as OleDbException
' Trata exceção de OleDb
Catch objEx as Exception
    ' Trata exceção genérica
End Try
```

As exceções são agrupadas hierarquicamente. A classe Exception (System) é a classe-base para todas as exceções. Todas as exceções são mapeadas em classes .NET.

Página de erro personalizada

Para especificar uma página de erro-padrão para uma aplicação ASP.NET, é necessário incluir a seguinte linha no arquivo Web.Config no diretório raiz da aplicação:

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

O atributo **mode** especifica quando a página de erro personalizada será exibida:

Valor	Descrição
On	A página de erro será exibida quando ocorrer um erro.
Off	A página de erro não será exibida (default).
Remoteonly	A página de erro será exibida somente para as estações, enquanto no servidor será exibida a página de erro default do ASP.NET que fornece informações de depuração.

Pode-se definir também páginas diferentes para cada erro, como a seguir:

```
<configuration>
<system.web>
<customerrors defaultredirect="paginaErroGeral.aspx" mode="On">
   <error statuscode="500" redirect="/errorpages/paginaerro500.aspx" />
   <error statuscode="404" redirect="/errorpages/paginaerro404.aspx" />
   <error statuscode="403" redirect="/errorpages/paginaerro403.html" />
</customerrors>
</system.web>
</configuration>
```

Além disso, é possível especificar páginas de erro individuais para cada página, utilizando o atributo ErrorPage da diretiva @Page no início da página:

```
<%@Page ErrorPage="/errorpages/paginanerro.aspx"%>
```

Trace

O trace é um recurso do ASP.NET para coleta de informações sobre a execução da página, como tempos de execução, objetos utilizados, cabeçalhos HTTP e variáveis do servidor. Essas informações podem ser exibidas na própria página ou em uma página separada. Pode-se também inserir comandos de trace nas páginas, da mesma maneira que o Response.write era utilizado nas versões anteriores. Porém, com o trace, não é necessário remover manualmente essas informações para que elas não fiquem visíveis para o usuário, basta simplesmente desabilitar o trace.

O objeto Trace (classe TraceContext) oferece métodos e propriedades para o rastreamento de execução de código. Pode-se utilizá-lo da seguinte forma em uma página ASP.NET:

Trace.Write (*categoria, mensagem***)** ou

Trace.Warn (*categoria, mensagem***)**

Parâmetro	Descrição
categoria	(String) Utilizada para agrupar ou identificar informações de rastreamento.
mensagem	(String) Mensagem a ser exibida.

As mensagem gravadas com Trace.Warn aparecerão em vermelho na página de trace.

Rastreamento no nível de página

O rastreamento no nível de página pode ser ativado por meio do atributo Trace da diretiva @Page:

<%@Page Language="*Vb***" Trace="***true***" %>**

O rastreamento no nível de página sobrepõe a configuração de rastreamento definida no arquivo Web.Config.

Quando a página é executada, uma tabela é adicionada ao rodapé dela contendo informações de rastreamento da página, como: Id da sessão, hora em que a solicitação foi processada, tempo de execução de cada parte da página (o que é útil para otimizações), árvore contendo os controles instanciados na página, coleção Cookies, cabeçalhos HTTP e coleção ServerVariables.

Essas mesmas informações estão disponíveis por meio do utilitário trace.axd quando o rastreamento no nível de aplicação está habilitado.

Rastreamento no nível de aplicação

Para habilitar o recurso de rastreamento para uma aplicação ASP.NET inteira, é necessário incluir a seguinte linha no arquivo Web.Config no diretório raiz da aplicação:

```
<trace enabled="false"
   requestLimit="10"
   pageOutput="false"
   traceMode="SortByTime"
   localOnly="true"
/>
```

Atributo	Descrição
Enabled	Habilita (true) ou desabilita (false)o rastreamento (default=False).
PageOutput	
True	Se a informação de rastreamento é exibida na própria página e na página trace.axd.
False	Se for exibida somente na página trace.axd (default).
RequestLimit	Número de solicitações a serem armazenadas no servidor com informações de rastreamento (default=10).
TraceMode	Indica a ordem de exibição das informações de rastreamento.
SortByTime	Exibe ordenando pela ordem de processamento (default).
SortByCategory	Exibe ordenando alfabeticamente e agrupada por categorias definidas pelo usuário.
LocalOnly	**true** se a página trace.axd está disponível somente no servidor; **false**, caso contrário (default).

Para acessar as informações de ratreamento, deve-se utilizar o utilitário trace.axd. Para acessá-lo basta digitar o caminho:

http://host/Nome da Aplicação/trace.axd

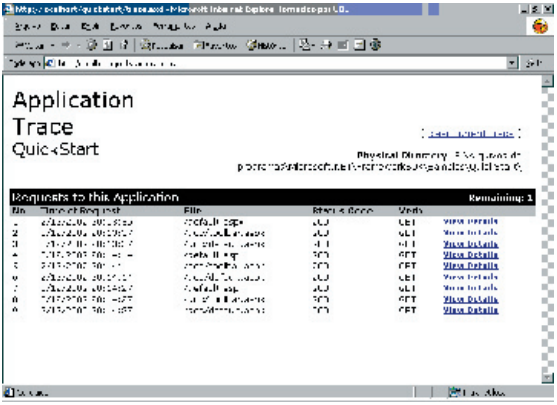

Figura 2.6 – Informações de rastreamento do utilitário trace.axd.

Debug

O Framework .NET oferece a ferramenta de debug CLR Debugger (\Microsoft.NET\FrameworkSDK\GuiDebug\DbgCLR.exe), que aproveita a natureza compilada das páginas ASP.NET e oferece recursos de depuração de código, como a utilização de "breakpoints" e a visualização do conteúdo de variáveis e expressões.

Para utilizar o CLR Debugger, deve-se seguir os seguintes passos:

1. Habilite o modo de debug da página a ser depurada, definindo o atributo Debug da diretiva @Page, como mostrado a seguir:

   ```
   <%@ Page Language="VB" debug=true %>
   ```

   Ou no arquivo Web.Config:

   ```
   <configuration>
      <compilation debug="true"/>
   </configuration>
   ```

2. Dentro do CLR Debugger, clique no menu **File...Open File** e selecione a página a ser depurada.
3. No menu **Tools**, clique em **Debug Processes** e selecione o processo Aspnet_wp.exe (marque a opção **Show system processes**).
4. Clique em **Attach** e em seguida **Close**.
5. Insira os "breakpoints" nos pontos desejados da página, clicando na margem esquerda da janela do CLR Deugger.
6. Execute a página ASP.NET a ser depurada a partir do browser. Quando um "breakpoint" for atingido, o debugger será chamado.

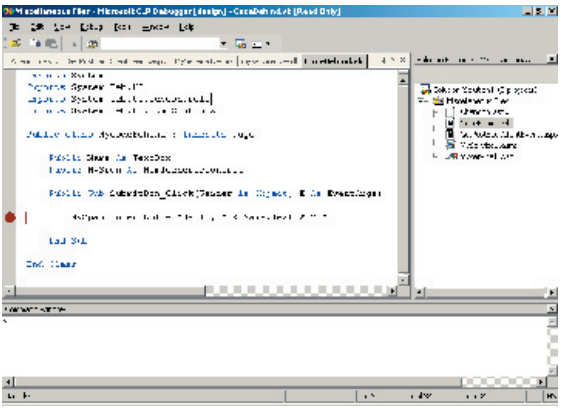

Figura 2.7 – Tela da ferramenta de debug do Framework .NET.

Cache

O cache é uma técnica utilizada para melhorar o desempenho de acesso a dados que são acessados com freqüência. O ASP.NET implementa o cache em diversos níveis. O código compilado da página é armazenado automaticamente em cache no servidor para melhorar o desempenho. Além disso, o ASP.NET suporta três tipos de cache de servidor: Cache de Saída, Cache de Fragmento e Cache de Dados.

Note que há duas localizações de cache de dados: do lado do cliente e do lado do servidor. O cache realizado do lado do cliente é realizado pelo browser. O cache do lado do servidor é gerenciado pelo servidor Web. Além disso, o conceito de cache não deve ser confundido com o armazenamento em buffer. O buffer é utilizado para armazenar uma saída temporariamente até que a execução da página esteja concluída.

Cache de saída

O cache de saída armazena a página inteira no cache e utiliza-o em solicitações futuras em vez de reprocessar a página solicitada. O cache de saída é feito por meio da diretiva @OutputCache.

O seguinte exemplo armazenará a página em cache por 30 segundos e, durante esse período, todas as solicitações a essa página serão servidas pelo cache.

```
<%@ Page Language="VB" %>
<%@ OutputCache Duration="30" VaryByParam="none" %>
<html>
<body bgcolor="#FFFFFF">
<p>Esta página foi gerada às: <strong><%= Now() %></strong></p>
</body>
</html>
```

Além disso, o parâmetro VaryByParam da diretiva @OutputCache permite especificar para quais parâmetros de querystring devem ser gerados no cache da página. Por exemplo, a diretiva a seguir indica ao ASP.NET para armazenar em cache versões diferentes da página com base no parâmetro de querystring "Nome":

```
<%@ OutputCache Duration="30" VaryByParam="Nome" %>
```

Cache de fragmento

O cache de fragmento permite armazenar partes de uma página em cache, como, por exemplo, o cabeçalho e rodapé da página. Para tal é necessário embutir o código a ser colocado em cache em um User Control e configurar suas opções de cache com a diretiva @OutputCache.

Cache de dados

O cache de dados pode ser utilizado para armazenar objetos que podem ser reutilizados nas páginas, melhorando o desempenho. Além disso, ele também fornece uma maneira simples de manter informação entre as páginas, porém deve-se atentar que o cache é global para a aplicação e a informação armazenada em cache estará visível a todos os usuários.

O cache pode ser acessado por meio da classe Cache, a qual fornece os métodos necessários para a manipulação dos objetos em cache. O exemplo a seguir ilustra a utilização do cache de dados:

```
<%@ Page Language="VB" %>
<script runat="server">
    Sub Page_Load(Sender As Object, E As EventArgs)
        Dim strTimestamp As String
        ' Recupera objeto armazenado no cache
        strTimestamp = Cache.Get("ValorExemplo")
        ' Verifica se o objeto ainda existe
        If strTimestamp Is Nothing Then
            ' Associa um novo valor
            strTimestamp = Now()
            ' Insere no cache – formas aceitáveis:
            '  Insert(String, Object)
            '  Insert(String, Object, CacheDependency)
            '  Insert(String, Object, CacheDependency, DateTime, TimeSpan)
            '  Insert(String, Object, CacheDependency, DateTime, TimeSpan, _
            ' CacheItemPriority, CacheItemRemovedCallback)
            Cache.Insert("ValorExemplo", strTimestamp, Nothing, _
                DateTime.Now.AddMinutes(.5), TimeSpan.Zero)
        End If
        lblTimestamp.Text = strTimestamp
    End Sub
</script>
<html>
<head>
<title>Exemplo de Cache de Dados ASP.NET</title>
</head>
<body bgcolor="#FFFFFF">
<p>
Esta página foi gerada às:
<strong><asp:label id="lblTimestamp" runat="server" /></strong>
</p>
</body>
</html>
```

Segurança

O ASP.NET trabalha em conjunto com o Internet Information Server (IIS) para dar suporte aos serviços de autenticação, autorização e personificação (Impersonation). Além disso, o ASP.NET suporta segurança baseada em Roles. As classes de segurança do Framework .NET são encontradas nos namespaces System.Web.Security e System.Web.Principal. O ASP.NET oferece também recursos de criptografia, assinatura digital, hashing e autenticação de mensagens. O namespace System.Security.Cryptography fornece as classes que implementam esses mecanismos.

Autenticação

A autenticação é a identificação do usuário que fez a solicitação, validando suas credenciais, que podem ser seu nome e sua senha. O ASP.NET suporta os seguintes tipos de autenticação:

Forms

Permite que aplicações forneçam sua própria página de "login" e realizem a verificação de credenciais. O ASP.NET redireciona a solicitação não autenticada para essa página. A página é submetida, a aplicação autentica a solicitação por meio do objeto FormsAuthentication (classe FormsAuthentication) e o sistema cria um cookie contendo as credenciais necessárias à autenticação do usuário. As solicitações subseqüentes do browser automaticamente incluirão o cookie.

A autenticação Forms é implementada da seguinte forma:

- Habilitar o acesso anônimo no IIS.
- Configurar a seção <authentication> e <authorization> no arquivo Web.Config. O acesso dos usuários anônimos não deve ser permitido:

```
<authentication mode="Forms">
   <forms name=".ASPXAUTH"  loginUrl="login.aspx"
   protection="All"
   timeout="30"
   path="/" />
</authentication>
<authorization>
   <deny users="?" />
</authorization>
```

Atributo	Descrição
LoginUrl	Solicitações não autenticadas serão redirecionadas para esta página.
Name	Nome do cookie de autenticação.
Path	Caminho do cookie de autenticação.
Protection	Opções de proteção dos dados do cookie. Este pode ser All, None, Encryption, Validation.
Timeout	Tempo de expiração do cookie de autenticação em minutos.

- Criar página de login, onde o usuário será validado, o cookie será gerado e a solicitação do usuário será redirecionada. Por exemplo, o seguinte código de autenticação poderia ser usado na página de login:

```
If FormsAuthentication.Authenticate(txtUserName.Value,txtUserPass.value) Then
  FormsAuthentication.RedirectFromLoginPage(txtUserName.Value, chkPersistCookie.Checked)
Else
  Response.Redirect("logon.aspx", false)
End If
```

  O método Authenticate do objeto FormsAuthentication retornará True se o nome do usuário e a senha combinarem com o usuário e a senha configurados no arquivo Web.Config.

Windows

Na autenticação Windows o IIS realiza a autenticação comunicando-se diretamente com o sistema operacional para validar as credenciais do usuário. A autenticação é repassada ao ASP.NET. A autenticação Windows requer o mínimo de codificação.

O IIS utiliza três tipos diferentes de autenticação:

Tipo	Descrição
Basic	O nome do usuário e a senha são transmitidos em cada solicitação. O IIS mapeia o nome do usuário e a senha para um conta no servidor Web, produzindo um token de acesso que é utilizado para verificar o acesso aos recursos por meio das listas de controle de acesso (ACL).
Digest	Semelhante à autenticação Basic, porém a senha do usuário não é transmitida como texto aberto. Requer browser que suporte autenticação Digest.
Integrada (NTLM)	O usuário não entra com seu nome e senha. Em vez disso, após o browser entrar em contato com o servidor, as informações criptografadas de nome do usuário e a senha que o usuário utilizou para se logar na estação são enviados para autenticá-lo no servidor. Requer que o browser seja o Internet Explorer.

Exemplo de configuração de autenticação Windows:

```
<authentication mode=" Windows" />
<authorization>
   <deny users="?" />
   <allow users= "*" />
</authorization>
```

Os caracteres especiais têm o seguinte significado:

Caractere	Descrição
*	Representa todos os usuários ou todos os grupos.
?	Significa acesso anônimo. Para a autenticação do tipo Windows, será interpretada como a conta de acesso anônima (IUSR_NOME_DO_COMPUTADOR).

Passport

Autenticação baseada no serviço Passport da Microsoft. Funciona de maneira semelhante à autenticação Forms, porém toda a funcionalidade é fornecida pelo Passport. Cria um cookie de autenticação que é utilizado para autorização. Pode ser utilizado quando se necessita de uma autenticação única entre aplicações ou sites. Mais informações em www.passport.com.

A autenticação Passport é implementada da seguinte forma:

1. Instalar o Passport SDK.
2. Registrar-se no serviço Passport Microsoft.
3. Configurar o arquivo Web.Config:

```
<authentication mode="Passport">
    <passport redirectUrl="internal|url" />
</authentication>
```

None

A autenticação None (default) é utilizada quando não há necessidade de implementar segurança para a aplicação. Oferece ganho de performance para a aplicação e permite implementar autenticação customizada.

Configuração:

```
<authentication mode="None" />
```

Autorização

Autorização é o processo que verifica se um usuário autenticado tem acesso a um determinado recurso. O ASP.NET oferece dois tipos de autorização: FileAuthorization (autorização no nível de diretório e arquivo), que é realizada pela classe FileAuthorizationModule e é ativada quando a autenticação é do tipo Windows. A classe FileAuthorizationModule é responsável por autorizar o acesso do usuário baseado nas listas de controle de acesso (ACLs); UrlAuthorization (autorização no nível de URL), realizada pela classe UrlAuthorizationModule, a qual é responsável por autorizar o acesso baseado na Url da solicitação. Os elementos deny e allow no Web.Config controlam quais usuários têm acesso a quais recursos.

Exemplo de configuração de UrlAuthorization:

```
<authorization>
    <allow roles="Admins" />
    <allow roles="WebUsers" />
    <deny users="*" />
</authorization>
```

Personificação

A personificação (Impersonation) permite ao ASP.NET executar páginas com a identidade do usuário que fez a solicitação, ou seja, o ASP.NET restringirá ou negará o aceso baseado nas permissões desse usuário. A personificação não é mais implementada por default no ASP.NET, como nas versões anteriores do ASP.

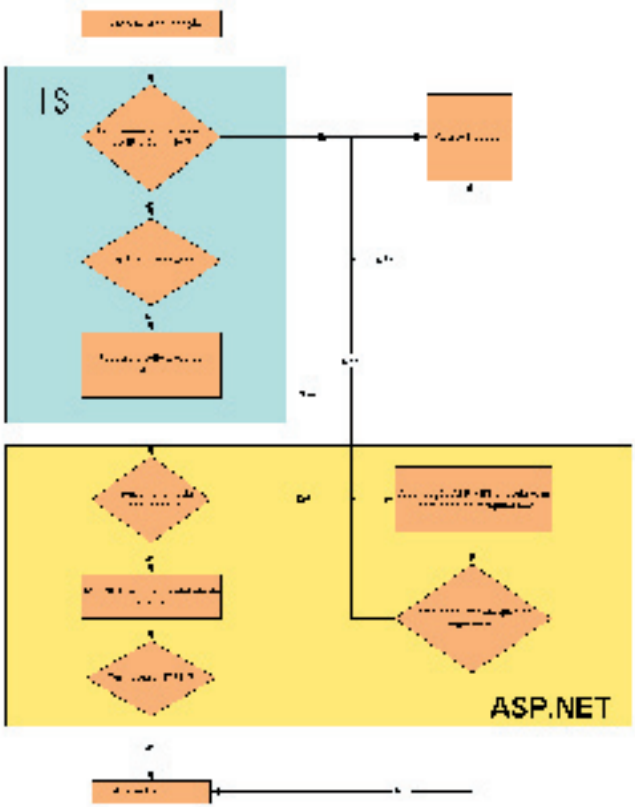

Figura 2.8 – Fluxo de segurança utilizando personificação.

Os seguintes passos são executados no atendimento a uma solicitação do usuário:

1. Um usuário solicita uma página .aspx ao IIS.
2. As credenciais do usuário são passadas ao IIS.
3. O IIS autentica o usuário e repassa o "token" de autenticação ao ASP.NET.
4. Com base no "token" de autenticação e nas configurações da aplicação (web.config), o ASP.NET decide se deve personificar um usuário. Se a personificação estiver habilitada, o ASP.NET assume a identidade do usuário que fez a solicitação e as permissões NTFS são verificadas. Caso a personificação não esteja habilitada, o ASP.NET executa-a com a identidade do IIS.
5. Se o acesso for permitido, o ASP.NET retorna à página solicitada por meio do IIS.

Exemplo de configuração de personificação:

```
<identity impersonate = "false | true" />
```

ou (personifica para um usuário determinado):

```
<idenity impersonate = "true" userName = "username" password = "password" />
```

Distribuição de uma aplicação ASP.NET

A distribuição de uma aplicação ASP.NET pode ser feita simplesmente por meio da cópia da estrutura de diretórios, contendo as páginas ASP.NET, os arquivos de configuração (Web.Config), o arquivo Global.asax e componentes (assemblies), para o local de destino. Não é necessário registrar um assembly, ou seja, nenhuma entrada no Registry do Windows deve ser criada. Para que um assembly esteja disponível para a aplicação, deve ser copiado para o diretório \bin na raiz da aplicação. Além disso, é possível fazer a cópia de um assembly sobre um assembly existente em uma aplicação em execução, mesmo que o próprio assembly esteja em uso. O Framework .NET possui um mecanismo que mantém uma cópia do assembly que está sendo utilizado, enquanto o assembly original não fica bloqueado para atualizações. Se o assembly for atualizado, novas solicitações farão uso do assembly atualizado, e as solicitações em execução ainda utilizarão a versão antiga do assembly, até que estas sejam concluídas.

O utilitário **GacUtil.exe**, que acompanha o Framework .NET, pode ser utilizado para listar, adicionar ou remover assemblies do cache global de assembly.