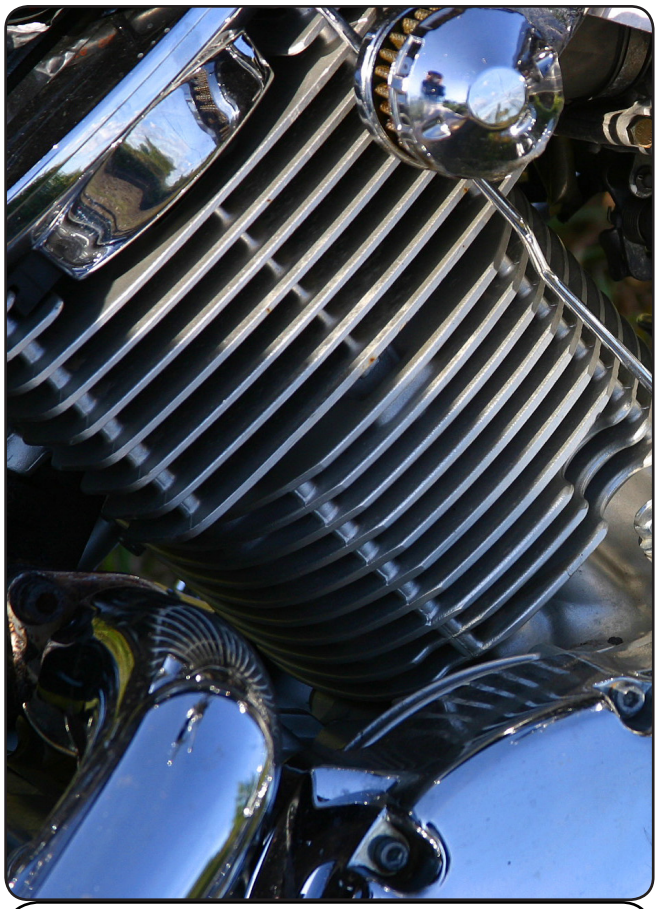
Nesta seção você encontra artigos para iniciantes em .net



Por dentro do Global.asax

Tratando um website como uma aplicação

De que se trata o artigo

O que é o Global.asax;
Eventos do Global.asax;
Aplicação prática: Tratando erros com global.asax.

Para que serve

Controlar eventos do web site, no nível da aplicação.

Em que situação o tema é útil

Aplicações web que precisam controlar eventos diversos, como o início da aplicação, o fim, o início de um request, ou um erro não gerenciado.

Thomas Alexander Semple

thomas.semple@t4w.com.br

27 anos, é engenheiro eletrônico com ênfase em telecomunicações e é líder de projetos da T4W, com mais de 10 anos de experiência, participa de projetos para empresas dos mais diversos segmentos. A T4W é uma empresa de tecnologia cuja área de desenvolvimento atende empresas de médio e grande porte em projetos corporativos e especiais. Além da .Net Magazine, Thomas Alexander Semple escreve para a Web Mobile e grandes portais de tecnologia.

Resumo do DevMan

O global.asax é capaz de tratar eventos da aplicação de forma global. Com ele é possível tratar erros ocorridos em toda aplicação, manipular respostas ao usuário, criar logs de acesso, iniciar eventos no início de uma sessão entre muitas outras funcionalidades.

Aplicações web estão cada vez mais parecidas com aplicações desktop, seja no modo de programar, seja na usabilidade cada vez mais avançada com a web 2.0. Quando programamos para web, não pensamos em projetos web como sendo uma aplicação, mas se observado, o IIS trata cada site como uma, através de seus diretórios virtuais. Estas aplicações podem ser reiniciadas, paradas, removidas, inicializadas assim como qualquer outra aplicação. Neste artigo veremos o que é o global.asax e como utiliza-lo de forma a tratar eventos que ocorrem no nível da aplicação. No exemplo prático, será feito um *log* de erro da aplicação web.

Por dentro do Global.asax

Quem já programou em ASP, seja o ASP 2.0, 3.0, já está acostumado em ter um arquivo que orbita por toda aplicação, o "global.asa". O ASP.NET possui uma versão mais atualizada do global.asa, o global.asax. Este arquivo, embora opcional nas aplicações web, pode trazer inúmeros benefícios se bem utilizado, e permitir rastrear uma série de informações inerentes ao site, como monitorar erros, criar um log de acesso, segurança, cache, sessão, entre outros. O global.asax trata respostas no

nível da aplicação e eventos da sessão ASP.Net, permitindo programar no lugar do *HttpApplication*, que representa a aplicação, implementando métodos e tratando eventos.

O arquivo deve ficar na raiz do projeto web. Por ser opcional, se o arquivo não existir na raiz, ele passa a assumir um comportamento "padrão" do ASP.Net. Caso o arquivo exista, a aplicação passa a utilizar os seus métodos. Dependendo do evento da aplicação, o global.asax é executado, e sua resposta pode ser enviada junto com a requisição http feita pelo usuário da aplicação.

Se o arquivo for alterado, o CLR (*Common Language Runtime*) detecta a alteração, reiniciando o aplicativo para que as novas ações possam ser executadas.

Na **Tabela 1** temos uma descrição dos principais eventos do global.asax. Dentre estes eventos, gostaria de destacar o *"Application_init"*. Este evento ocorre quando a aplicação inicializa, permitindo que você invoque alguns métodos de verificação de ambiente, por exemplo.

O *"Session_Start"* é acionado sempre que um novo usuário visita a aplicação, e o "Session_End" sempre que a sessão de um usuário encerra. Estes métodos permitem uma interação mais próxima ao usuário como, por exemplo, criar um contador de usuários online no sistema em um dado momento. No exemplo prático deste artigo, vamos utilizar o *"Application_Error"*, que é acionado sempre que um erro não tratado é levantado no sistema. Permitindo também um uso imediato para qualquer aplicação web. No exemplo, vou mostrar como interagir com o global.asax através de classes personalizadas. O mesmo conceito poderá ser estendido para qualquer dos eventos da **Tabela 1**.

Requisições HTTP

Quem já desenvolveu alguma aplicação para Web, por mais simples que seja, está acostumado com os termos *POST* e *GET*. Quando utilizamos o método *GET*, os dados são enviados através de *Query String*, ou seja, junto com o endereço *http* da página. Você já deve ter observado em endereços *http* que ao final do arquivo tem um ponto de interrogação seguido de outras informações, como por exemplo: *meusite.com.br/pagina.aspx?verpagina=QuemSomos*. Neste caso o texto até *pagina.aspx* é o endereço, a *Query String* possui uma variável "verpagina" que possui o valor "QuemSomos". No *Asp.Net* podemos recuperar esta informação com o comando *"request.querystring("verpagina")"*.

O Método *POST* permite que os dados sejam enviados juntos com o formulário. Embora o *.NET framework* abstraia muito destes conceitos, em *ASP.NET* é possível recuperar um valor de um campo do formulário utilizando o *request.form*("nomedocampo"). Este conceito é importante, pois o global.asax pode interagir com estas requisições.

Preparando o ambiente

Vamos agora criar uma aplicação para ver como utilizar o global.asax na prática. Nesta aplicação vamos criar um log, em

Método	Descrição
Application_Init	Evento invocado quando a aplicação inicializa, ou é acessada pela primeira vez. É invocada pela instância dos objeto HttpApplication.
Application_Disposed	Evento é invocado quando a aplicação é destruída. É normalmente utilizada para liberar recursos da memória.
Application_Error	Quando um erro não tratado ocorre, este evento é invocado.
Application_Start	Quando a primeira instância da HttpApplication é criada, este método é invocado, permitindo que objetos aqui criados sejam acessados por todas as instancias de HttpApplication.
Application_End	Invocado quando a ultima instancia da classe HttpApplication é destruída. É invocado somente uma vez durante o ciclo de vida da aplicação.
Application_BeginRequest	É invocado quando há um recebimento de requisição para a aplicação. É o primeiro evento invocado por requisição.
Application_EndRequest	O último evento invocado em uma requisição da aplicação.
Application_PreRequestHandlerExecute	Invocado antes da aplicação ASP.NET inicializa a execução de eventos, seja de uma página, ou de um web service.
Application_PostRequestHandlerExecute	Invocado após a aplicação ASP.NET finalizar a execução de eventos.
Applcation_PreSendRequestHeaders	Invocado antes da aplicação ASP.Net enviar o cabeçalho da requisição HTTP para o cliente.
Application_PreSendContent	Invocado antes da aplicação ASP.NET enviar o conteúdo da requisição para o cliente.
Application_AcquireRequestState	Invocado quando a aplicação ASP.NET recebe o estado referente à requisição atual (*current state*).
Application_ReleaseRequestState	Invocado quando a aplicação ASP.NET completa a execução de todos os eventos.
Application_AuthenticateRequest	Evento invocado quando o módulo de segurança (*security module*) estabeleceu para o usuário uma identidade válida. Nesta etapa sua credencial foi validada.
Application_AuthorizeRequest	Invocado quando o modulo de segurança (*security module*) fez a verificação que o usuário pode acessar determinado recurso da aplicação.
Session_Start	Invocado quando um novo usuário visita a aplicação ASP.NET.
Session_End	Evento invocado quando a sessão de um usuário expira, seja por timeout, ou por desconexão.

Tabela 1: Principais eventos do Global.asax

XML de todos os erros ocorridos na aplicação web. Quando um erro ocorrer na aplicação web, seja em uma página web "*.aspx*", seja em um *web service*, ou até em um *handler* genérico, este erro pode ser tratado pelo global.asax. Vamos aproveitar este recurso e criar um modo de "logar" estes erros. Nem preciso falar o quão importante é poder rastrear os erros ocorridos na aplicação, seja para uma manutenção ativa, seja para dar um melhor suporte ao usuário. Através deste simples exemplo, vou mostrar como interagir com o global.asax, de uma forma *multi-camada*, ou seja, separando a lógica em uma classe separada.

Neste projeto utilizei o Visual Studio 2005. Pode ser utilizada qualquer versão do Visual Studio, inclusive as edições Express e 2008. Para baixar a versão express do Visual Studio acesse *www.microsoft.com/Express/*.

Abra o Visual Studio e crie uma aplicação web, selecionando "File → NewWeb Site", conforme **Figura 1**. Uma caixa de diálogo será apresentada perguntando o tipo de aplicação que deseja criar. Selecione o tipo como *"ASP.NET Web Site"*, a linguagem *Visual Basic* e informe a pasta onde o projeto será salvo na sua máquina (**Figura 2**). Como o intuito deste exemplo é apresentar alguns conceitos do global.asax, caso queira utilizar outra linguagem compatível com a plataforma .Net, sinta-se à vontade.

Agora que você tem o projeto criado, adicione dois arquivos, clicando com o botão direito do mouse, em cima do projeto web, na caixa de ferramentas *"Solution Explorer"*, conforme **Figura 3**. Selecione a opção "Add New Item...". Adicione o arquivo de Global.asax, selecionando a opção de arquivo "Global Aplication Class". Deixe o nome como "Global.asax", conforme **Figura 4**. Note que no projeto web, na raiz, o arquivo global.asax foi adicionado. Se você abrir este arquivo notará que diversas linhas de códigos já foram adicionadas. Por enquanto não altere nada. Adicione agora um segundo arquivo, agora de classe, conforme **Figura 5**, chamado "logErro.vb". Caso sua aplicação ainda não tenha uma pasta chamada "App_code", o Visual Studio perguntará se você quer adicioná-la. Selecione a opção que "sim". Esta pasta contém todas as classes do projeto web, que não estão ligadas a nenhuma outra página.

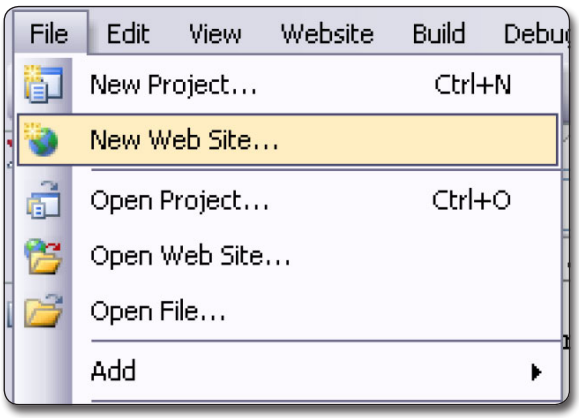


Figura 1: Crie um novo Web site

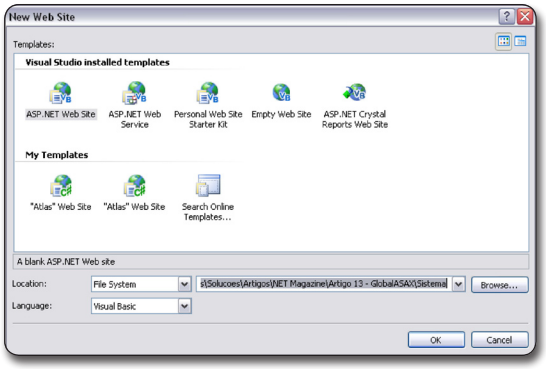


Figura 2: Selecione a pasta, o tipo como ASP.NET Web Site e a linguagem Visual Basic

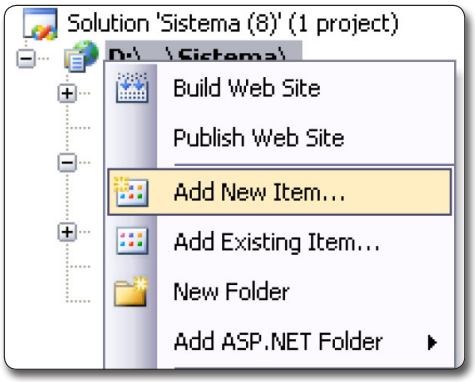


Figura 3: Adicionando um novo ítem.

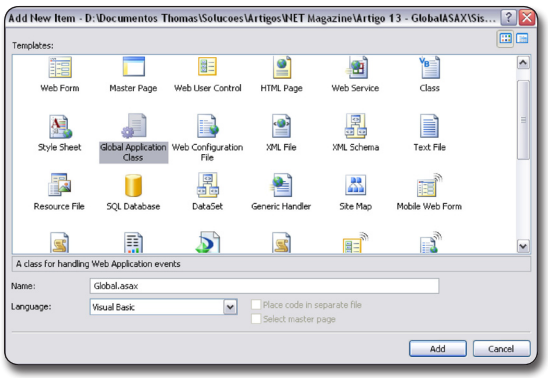


Figura 4: Adicione o arquivo "Global.asax" no projeto.

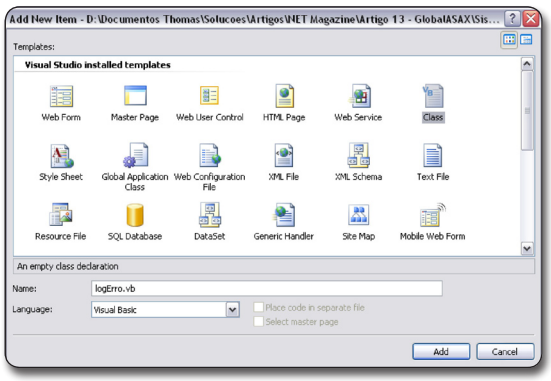


Figura 5: Adicione uma classe chamada logErro.vb .

Classe de erro

Abra o arquivo "logErro.Vb", e adicione o código da **Listagem 1**. Na linha 01, temos o "*Imports Microsoft VisualBasic*", que já vem automaticamente na criação do arquivo. Isto indica que o código foi escrito utilizando recursos da linguagem *Visual Basic*.

Na linha 03 é definida a criação da classe "logErro". Na linha 05, temos uma linha de marcação, definida como "#Region Proriedades". A *region* serve para agrupar trechos de códigos, e facilitar a visualização por parte do desenvolvedor, pois o Visual Studio permite ocultar as linhas, utilizando o "+", ou "-", ao lado das linhas. Eu particularmente gosto deste recurso, pois permite focar e organizar o que esta sendo desenvolvido. Vamos agora criar as propriedades da classe. A primeira propriedades a ser criada é a "MsgErro". Esta propriedade receberá a mensagem de erro. Todo erro que ocorre, possui um "trace", ou seja, possui um rastro dos eventos executados antes de ocorrer o erro. Esta informação será atribuída à propriedade "Trace", definida na linha 25. A próxima propriedade a ser criada é a "URL". Esta propriedade informará o endereço onde ocorreu o erro da aplicação. Como informação adicional, vamos salvar a data/hora do erro e o IP do usuário, definidos respectivamente nas linhas 54 e 67 desta mesma listagem.

Na **Listagem 2**, vamos implementar o método responsável por salvar este erro. O objetivo deste método é salvar todos os valores recebidos pelas propriedades da classe em um arquivo .XML. Neste método, foi utilizado um exemplo bem didático para criação do XML. Você pode utilizar a forma que estiver mais acostumado, ou inclusive utilizar tecnologias mais recentes como o "LINQ to XML", ou "XML Literals".

Nota

Prefira salvar o erro em um arquivo texto estruturado, por exemplo, um arquivo XML. Se for salvar o log em uma base de dados, e a base de dados cair, a aplicação levantará um erro, e tentará salvar o log na base de dados, que apresentou o erro, e levantará outro erro.

Além de você perder o log deste período, a aplicação poderá apresentar loop indesejáveis, podendo inclusive derrubar seu servidor dependendo do número de acessos. Quando o erro é salvo em um arquivo texto, a chance da aplicação não conseguir salvar o erro é quase zero, a não ser que o caminho esteja incorreto (o que pode ser tratado), ou o disco rígido der algum erro crítico. Mesmo se a base de dados cair, o erro que ocasionou o problema é salvo.

Crie o método "salvarLog", conforme linha 113. Está *sub* recebe como parâmetro o caminho completo do arquivo XML onde o log será salvo. Note que o caminho deverá ser o caminho físico, como por exemplo "c:\aplicacao\erros.xml". Crie uma variável "dt" como um *data.datatable*, conforme linha 116. Datatable é uma classe do ADO.NET que representa uma tabela. Assim como uma tabela de banco de dados, o datatable possui linhas (*datarow*) e colunas (*datacolumn*). Vamos atribuir o nome da tabela como "*Log*", conforme linha 117. Para cada propriedade criamos uma coluna, conforme código da linha 118 a 122, com o comando "dt. Columns.Add("MsgErro")", por exemplo, estamos atribuindo uma coluna chamada "*MsgErro*" ao datatable "dt".

Depois de criar as colunas, vamos verificar se o arquivo existe (linha 125). Se o arquivo existir, ou seja, provavelmente já tenha ocorrido um erro na aplicação, todas as linhas são lidas, para manter as informações. Caso o arquivo não exista, ele então é criado, em branco, utilizando o comando de "*System.IO.File*" chamado "*WriteAllText*", passando o caminho e *string.empty* (vazio), como parâmetro.

Listagem 1: Classe de log de erro, arquivo app_code/logErro.vb

```
(Parte a: Propriedades)
  1 Imports Microsoft.VisualBasic
  2
  3 Public Class logErro
  4
  5 #Region "Propriedades"
  6
  7     ''' <summary>
  8     ''' Propriedade que recebe a
  9     ''' mensagem de erro
 10     ''' </summary>
 11     Private m_sMsgErro As String
 12     Public Property MsgErro() As String
 13         Get
 14             Return m_sMsgErro
 15         End Get
 16         Set(ByVal value As String)
 17             m_sMsgErro = value
 18         End Set
 19     End Property
 20
 21     ''' <summary>
 22     ''' Propriedade que recebe os
 23     ''' detalhes do erro
 24     ''' </summary>
 25     Private m_sTrace As String
 26     Public Property Trace() As String
 27         Get
 28             Return m_sTrace
 29         End Get
 30         Set(ByVal value As String)
 31             m_sTrace = value
 32         End Set
 33     End Property
 34
 35     ''' <summary>
 36     ''' Propriedade que recebe o caminho
 37     ''' onde ocorreu o erro
 38     ''' </summary>
 39     Private m_sURL As String
 40     Public Property URL() As String
 41         Get
 42             Return m_sURL
 43         End Get
 44         Set(ByVal value As String)
 45             m_sURL = value
 46         End Set
 47     End Property
 48
 49     ''' <summary>
 50     ''' Propriedade que recebe a data e
 51     ''' hora de quando ocorreu o erro
 52     ''' </summary>
 53     ''' <remarks></remarks>
 54     Private m_dData As DateTime
 55     Public Property Data() As DateTime
 56         Get
 57             Return m_dData
 58         End Get
 59         Set(ByVal value As DateTime)
 60             m_dData = value
 61         End Set
 62     End Property
 63
 64     ''' <summary>
 65     ''' Retorna o IP do usuário
 66     ''' </summary>
 67     Private m_sIP As String
 68     Public Property IP() As String
 69         Get
 70             Return m_sIP
 71         End Get
 72         Set(ByVal value As String)
 73             m_sIP = value
 74         End Set
 75     End Property
 76 #End Region
<...>
```

Listagem 2: Classe de log de erro, arquivo app_code/logErro.vb (Parte b: Métodos)

```
<...>
#Region "Métodos"

    ''' <summary>
    ''' Construtor da classe
    ''' </summary>
    Public Sub New()
        ' Limpo os valores, como
        ' uma boa prática
        limpar()
    End Sub

    ''' <summary>
    ''' Método que limpa os
    ''' valores das propriedades
    ''' </summary>
    Public Sub limpar()
        m_sMsgErro = String.Empty
        m_sTrace = String.Empty
        m_sURL = String.Empty
        m_dData = Date.MinValue
        m_sIP = String.Empty
    End Sub

    ''' <summary>
    ''' Salva o log em um .xml
    ''' </summary>
    ''' <param name="sCaminhoCompleto">
    ''' Caminho completo do arquivo de log.
    ''' </param>
    ''' <remarks>
    ''' Fique a vontade para implementar este método
    ''' para salvar em um banco de dados. Ou usando o modo
    ''' que achar mais adequado para salvar seu XML.
    ''' </remarks>
    Public Sub salvarLog(ByVal sCaminhoCompleto As String)

        ' Exemplo didático de como criar uma estrutura de tabela
        Dim dt As New Data.DataTable
        dt.TableName = "Log"
        dt.Columns.Add("MsgErro")
        dt.Columns.Add("Trace")
        dt.Columns.Add("URL")
        dt.Columns.Add("Data")
        dt.Columns.Add("IP")

        ' Leio o XML
        If System.IO.File.Exists(sCaminhoCompleto) Then
            ' Se o arquivo existir, os valores são lidos
            dt.ReadXml(sCaminhoCompleto)
        Else
            ' Se o arquivo não existir, ele é criado
            System.IO.File.WriteAllText(sCaminhoCompleto,
             String.Empty)
        End If

        ' Crio uma nova linha
        Dim dr As System.Data.DataRow = dt.NewRow

        ' Atribuo os valores para a linha
        dr("MsgErro") = m_sMsgErro
        dr("Trace") = m_sTrace
        dr("URL") = m_sURL
        dr("Data") = m_dData.ToString("dd/MM/yyyy HH:mm:ss")
        dr("IP") = m_sIP
        ' Adiciono a linha para a tabela
        dt.Rows.Add(dr)
        ' Salvo o arquivo
        dt.WriteXml(sCaminhoCompleto)
        ' Limpo o datatable da memória
        dt.Dispose()
    End Sub
#End Region
End Class
```

Uma nova linha de registro é então criada, e o valor de cada propriedade é atribuído à sua respectiva coluna do datatable. Por fim, o arquivo é salvo, e o *datatable* é liberado da memória utilizando o método *"dispose"*. Note que com poucas linhas de código conseguimos escrever um XML, com um log do erro. Precisamos agora receber os valores corretamente da aplicação, através do global.asax.

Editando o Global.asax

Agora que temos a classe de erro pronta, vamos criar o código do global.asax que enviará as informações para a classe. Como vimos anteriormente neste artigo, o arquivo global.asax é responsável por tratar eventos globais da aplicação. Não está atrelada a nenhuma página .aspx, nem a nenhum *web service* específico. O global.asax orbita em torno de toda a aplicação. Na **Listagem 3** temos o código para o arquivo global.asax. É criada uma variável interna "m_sCaminhoArquivoErro". Esta variável receberá o caminho físico do arquivo. Estou salvando o arquivo na pasta *"App_data"*. Esta pasta é padrão do ASP.NET. Ela é protegida, ou seja, o usuário não consegue fazer download do arquivo passando o endereço HTTP. Se o arquivo fosse salvo em outra pasta não protegida, o usuário poderia fazer download do seu arquivo de erro e usá-lo para fins maliciosos.

Como o Visual Studio cria uma séria de métodos padrão do global.asax automaticamente, você pode apagá-los, deixando apenas o de tratamento de erro *"Application_Error"*. No início deste método é atribuído o endereço do arquivo, utilizando o *"Server.MapPath(".")"*. Este método retorna o caminho físico da aplicação. Em Seguida, concateno com a pasta segura "\App_data\", e o nome do arquivo "erros.xml". Certifique-se que a pasta *"App_data"* existe em sua aplicação web, caso não exista, crie.

Nota

O ASP.NET 2.0 introduziu o conceito de pastas especiais, que levam nomes pré-definidos, que ficam abaixo do diretório do seu Web Site. Estas pastas são:

- App_Data: criada por padrão, serve para armazenar arquivos de dados como bancos de dados locais (SQL Server, ou mesmo XML);
- App_Code: usada para armazenar classes que não estejam vinculadas à páginas, como classes de negócio ou classes que contém métodos utilitários;
- App_GlobalResources: todas as páginas, user controls etc. podem usar recursos compartilhados armazenados de forma global;
- App_LocalResources: semelhante à pasta acima, válida a nível de página;
- App_Browsers: utilizada pelo ASP.NET para identificar diferentes tipos de browsers e suas capacidades;
- App_Themes: utilizada para armazenar temas e skins ou arquivos CSS em uma aplicação que usa o novo recurso de temas.

Em Seguida uma variável de *logErro* é criada. Observe que *"logErro"* é a classe criada na **Listagem 1**. Os valores de cada propriedade são atribuídos, como Data, IP, Mensagem de Erro, Trace, e URL. Note que para recuperar as informações de erro utilizamos *"Server.GetLastError"*. Este método nos retorna inúmeras outras informações do erro. Após salvar o log, invocando o método "salvarLog" criado na classe *logErro*, e passado o caminho do arquivo como parâmetro, redireciono o usuário para uma página genérica de erro. Esta página é criada na **Listagem 4.** Note que embora o usuário visualize uma página genérica de erro, o programador consegue saber exatamente o

que ocorreu com o erro, e corrigir o problema. A página amarela de erro padrão da aplicação ASP.NET geralmente assusta o usuário, utilizando uma página personalizada, o usuário fica um pouco mais tranqüilo. Outra grande vantagem é manter a segurança do sistema. Às vezes o erro pode revelar alguma instrução SQL ou algo parecido, permitindo que usuários maliciosos tirem proveito desta falha.

Listagem 3: Global.asax, tratamento de erro.

```
 1 <%@ Application Language="VB" %>
 2
 3 <script RunAt="server">
 4
 5 ' Variável que recebe o caminho do erro
 6 Private m_sCaminhoArquivoErro As String = String.Empty
 7
 8 ''' <summary>
 9 ''' evento que ocorre quando acontece um erro na aplicação
10 ''' </summary>
11 ''' <remarks></remarks>
12 Sub Application_Error(ByVal sender As Object, ByVal e As
    EventArgs)
13     ' Na inicialização, verifica se o arquivo xml existe
14     m_sCaminhoArquivoErro = Server.MapPath(".") & "\
        App_data\erros.xml"
15
16     Dim oErro As New logErro
17
18     ' Recebe a data
19     oErro.Data = Date.Now
20
21     ' Recebe o IP do usuário que recebeu o erro
22     oErro.IP = Request.UserHostAddress
23
24     ' Recebe a mensagem de erro
25     oErro.MsgErro = Server.GetLastError.Message.ToString
26
27     ' Recebe o stack do erro
28     oErro.Trace = Server.GetLastError.StackTrace.ToString
29
30     ' Recebe o caminho da página que gerou o erro
31     oErro.URL = Request.Url.ToString
32
33     ' Salva o log em um XML
34     oErro.salvarLog(m_sCaminhoArquivoErro)
35
36     ' Redireciona para uma página de erro
37     Response.Redirect("erroGenerico.aspx")
38
39 End Sub
40
41 </script>
```

Por fim, vamos criar uma página genérica de erro. Esta página será exibida sempre que ocorrer um erro para o usuário. Adicione ao seu projeto o arquivo "erroGenerico.aspx". Este arquivo pode ser personalizado conforme sua necessidade. Pode por exemplo estar contido dentro de uma master.page, com seu layout padrão. Tenha cuidado apenas para que essa página não gere um erro também, ou a aplicação entrará em looping, chamando a página de exibição do erro e voltando a ela sem parar.

Nesta página, é definido um título, utilizando "<h1>" do HTML, um subtítulo com "<h2>" e uma mensagem de erro, com informações sobre o que o usuário deve fazer. Note que é bem simples, e não interage com nenhum recurso adicional, principalmente banco de dados. Isto é importante, pois se a base de dados cair, não gera mais um erro em seqüência, como já foi dito. Rode a aplicação, clicando no botão "*play*" do Visual Studio, ou pressionando "F5". Uma tela do seu navegador padrão aparecerá com a tela defaul.aspx, que provavelmente não possui nada. Podemos simular um erro da aplicação, chamando uma página que não existem, como por exemplo *http://localhost:XXXX/Sistema/PaginaQueNaoExiste.aspx*, onde "XXXX" é a porta definida pelo Visual Studio para debug do sistema.

Chamando uma página que não existem, você deve visualizar uma página similar a **Figura 6**. Ao acessar a pasta /App_data, um arquivo XML deve ter sido criado. Na **Listagem 5**, temos o XML gerado com o log do erro. Note que o primeiro nó possui o nome de "Log", assim como o nome dado à tabela do *datatable* na classe da **Listagem 1**. No nível abaixo, há um nó XML para cada propriedade que criamos na classe de erro, como por exemplo, "MsgErro", "Trace", "URL", "Data" e "IP". Se outros erros ocorrem no sistema, eles serão salvos da mesma maneira, tendo um nó de log para cada erro.

IHTTPModule

Caso você tenha se interessado pelo Global.asax, um bom tópico para estudo é a implementação do IHTTPModule, o módulo Http chamado sempre que houver uma requisição em seu sistema Web. O IHttpModule é geralmente utilizado para implementar uma camada de Segurança, Log de acesso ou cabeçalhos/rodapés personalizados, e pode ser encapsulado em uma DLL a parte, permitindo que você re-aproveite facilmente seu código, sem a necessidade de ficar copiando e colando, além de dar mais segurança ao seu código fonte.

Listagem 4: Página de erro genérico, em erroGenerico.aspx

```
<%@ Page Language="VB" AutoEventWireup="false"
CodeFile="erroGenerico.aspx.vb"
Inherits="erroGenerico" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Erro de sistema</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h1>
                Ocorreu um erro</h1>
            <h2>
                Por favor, tente novamente</h2>

                Sua mensagem de erro vai aqui.....
        </div>
    </form>
</body>
</html>
```

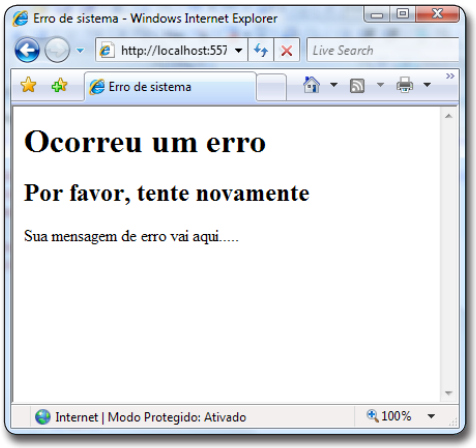


Figura 6: Tela de erro padrão que o usuário receber.

Listagem 5: Exemplo do XML Criado quando um erro ocorre no sistema

```
1 <?xml version="1.0" standalone="yes"?>
2 <DocumentElement>
3   <Log>
4     <MsgErro>The file '/Sistema/testeerro.aspx' does not
        exist.</MsgErro>
5     <Trace>  at System.Web.UI.Util.CheckVirtualFileExists(
       VirtualPath virtualPath)
6   at System.Web.Compilation.BuildManager.GetVPathBuildRe
     sultInternal(VirtualPath virtualPath, Boolean noBuild,
      Boolean allowCrossApp, Boolean allowBuildInPrecompile)
7   at System.Web.Compilation.BuildManager.GetVPathBuildResult
     WithNoAssert(HttpContext context, VirtualPath
      virtualPath, Boolean noBuild, Boolean allowCrossApp,
       Boolean allowBuildInPrecompile)
8   at System.Web.Compilation.BuildManager.GetVirtualPathObject
     Factory(VirtualPath virtualPath, HttpContext context,
      Boolean allowCrossApp, Boolean noAssert)
9   at System.Web.Compilation.BuildManager.CreateInstanceFrom
     VirtualPath(VirtualPath virtualPath, Type
      requiredBaseType, HttpContext context, Boolean allow
       CrossApp, Boolean noAssert)
10  at System.Web.UI.PageHandlerFactory.Get
     HandlerHelper(HttpContext context, String requestType,
      VirtualPath virtualPath, String physicalPath)
11  at System.Web.UI.PageHandlerFactory.System.Web.
     IHttpHandlerFactory2.GetHandler(HttpContext
      context, String requestType, VirtualPath virtualPath,
       String physicalPath)
12  at System.Web.HttpApplication.MapHttpHandler(HttpContext
     context, String requestType, VirtualPath path, String
      pathTranslated, Boolean useAppConfig)
13  at System.Web.HttpApplication.MapHandlerExecutionStep.
     System.Web.HttpApplication.IExecutionStep.Execute()
14  at System.Web.HttpApplication.ExecuteStep(IExecutionStep
     step, Boolean&amp; completedSynchronously)</Trace>
15    <URL>http://localhost:3804/Sistema/testeerro.aspx</URL>
16    <Data>21/08/2008 11:30:52</Data>
17    <IP>127.0.0.1</IP>
18  </Log>
19 </DocumentElement>
```

Conclusão

Quando queremos utilizar recursos que abrangem toda a aplicação, uma boa opção é o global.asax. Neste artigo foi apresentado modo de criar um log dos erros da aplicação. Gostaria de propor-lhe mais alguns desafios, como salvar mais detalhes do erro, e criar mecanismos para você receber notificações ativa deste erro.

Mais um desafio será o de criar um módulo de gerenciamento dos erros, para que você possa visualizar em uma *GridView* os erros, e marcar aqueles que já foram solucionados.

Apesar do exemplo prático focar em apenas um método do global.asax, ele abrangeu boa parte da teoria para que você possa utilizar outros métodos e tirar proveito de mais recursos globais, como o início de uma requisição ou da própria aplicação, onde você também pode tomar ações.

Criar aplicações robustas em ASP.Net é cada vez mais importante e entender bem as bases da plataforma .NET é essencial para que se tenha um bom desenvolvimento. ●

Dê seu feedback sobre esta edição!

A .NET Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/netmagazine/feedback