**参考：<http://www.cnblogs.com/lizhenlin/p/5487917.html>**

[**ASP.NET Global.asax详解**](http://www.cnblogs.com/lizhenlin/p/5487917.html)

最近在研究bbsmax的代码，但是一直不知道入口在哪里，然后就对各个文件分析了，然后终于在对global.asax文件查看的时候看到Application\_BeginRequest（这是程序运行在代码层面能捕捉到的起点）才明白入口，所以现在补救下这方面的知识。以下是转载的文档：

文档来源：<http://club.topsage.com/thread-485397-1-1.html>

global.asax是一个文本文件，它提供全局可用代码。这些代码包括应用程序的事件处理程序以及会话事件、方法和静态变量，Application\_Start/Seesion\_Start/Application\_BeginRequest/Application\_AuthenticateRequest/Application\_Error/Session\_End/Application\_End。有时该文件也被称为应用程序文件。  
global.asax 文件中的任何代码都是它所在的应用程序（web）的一部分。每个应用程序在其根目录下只能有一个global.asax文件。然而，这个文件是可选的。如果没有global.asax文件，应用程序将对所有事件应用由 HttpApplication类提供的默认行为,这个Global也是继承自HttpApplication。

提示：经典ASP有一个与global.asax类似格式和构造的，名为global.asa的文件。实际上，如果将一个正在运行的global.asa文件代码复制到global.asax中，应用程序同样可以运行。  
当应用程序运行的时候，global.asax的内容被编译到一个继承自HttpApplication类的类中。因此，HttpApplication类中所有的方法、类和对象对于应用程序都是可用的。

CLR 监控着global.asax的变化。如果它察觉到这个文件发生了改变，那么将自动启动一个新的应用程序复本，同时创建一个新的应用程序域。原应用程序域当前正在处理的请求被允许结束，而任何新的请求都交由新应用程序域来处理。当原应用程序域的最后一个请求处理完成时，原应用程序域即被清除。这有效的保证了应用程序可以重新启动，而不被任何用户察觉。

为防止应用程序用户下载应用程序而看到源代码，ASP.NET缺省配置为阻止用户查看global.asax的内容。如果有人在浏览器输入以下URL：

http://localhost/progaspnet/Global.asax  
这将会收到一个403（禁止访问）错误信息或者类似的信息如：  
This type of page is not served。

web.config和Global.asax都可以实现hotfix

提示：简单而言，web.config文件与global.asax有些类似的地方。如果这个文件被更改，应用程序将自动“重启”。同样，也不可能在浏览器中查看web.config文件。  
Global.asax文件从外观和结构上与页面文件（.aspx）相似。它可以有一个或多个部分，简要

描述如下：  
l 指令(如<%@ Application Language="C#" %>)  
l 脚本块（如：<script runat="server">some codes</script>）  
l Object声明(如：void Application\_Start(object sender, EventArgs e){ //do some stuff})

正如Web页和Web服务能够使用代码隐藏功能，global.asax同样也可以。然而，与Web页和Web服务的条件有所不同，VS2005默认状态下不对global.asax使用代码隐藏功能。  
提示：Visual Studio 2005预览版默认对global.asax使用代码隐藏模型。目前仍然支持代码隐藏，但不是默认使用。

为了对global.asax使用代码隐藏技术，可使用位于该文件头部的Application指令（类似于页面文件的Page指令，下一节将详细介绍）的Inherits属性，该属性指向global.asax.cs中的代码隐藏类。

同时，也有一个CodeBehind属性用来指向代码隐藏文件(me:一般为aspx对应的cs文件)。然而，如果它指向的是一个位于App\_Code文件夹以外的位置，那么必须对这个类文件进行手动编辑。

通过右键单击解决方案资源管理器中的网站或者单击网站菜单，然后选择“Add New Item...”，接着选择全局应用程序类，可以为Web应用程序添加一个global.asax文件。保留默认名称global.asax。

me:在新版的vs中可以直接创建一个global.aspx.cs的文件

VS2005将创建一个如示例18-1所列的文件。模板中包括对以下5个事件的空白声明：Application\_Start、Application\_End、Session\_Start、Session\_End和Applica- tion\_Error。

示例18-1：global.asax模板  
< %@ Application Language="C#" %>  
在示例18-2所列举的global.asax文件中，为应用程序状态设置了一些值，同时，在每次应用程序启动(Application\_Start)的时候向日志文件写一个条目。为了使用这个示例，需要保证ASP.NET帐户对根目录c:\具有写入权限（在产品系统中不推荐）。

示例18-2：global.asax示例  
< %@ Application  Language="C#"%>  
指令  
与Web页和Web服务文件相比，global.asax可以以多个指令作为开始。这些指令在处理ASP.NET文件时指定应用程序编译的设置。与Page指令相比，Application指令可接受一个或者多个具有字典结构的属性/值对。此处支持三个指令：Application、Import和Assembly。  
Application  Application指令设置编译器的应用程序专用属性。以下是一个Application指令示例：  
< %@ Application  Language="C#" Inherits="WebServiceConsumer.Global"  
Description="A sample application" %>

Language 属性可以设置为任何一种标准语言名称：VB、C#、JS、或VJ#，它们分别对应VB2005、C#、JScript.NET或J#。（可以使用任何一种支持.NET平台的第三方语言）默认值为C#。此处的Language设置的是语言专门用于global.asax文件，而非其他应用程序代码文件。例如，可以完全合法地在global.asax文件中使用C#，在.aspx文件中使用VB2005，相反亦然。

Inherits属性指定所继承类名，具有代表性的如代码隐藏文件中的类。  
Description属性接受对应用程序的文本描述，而分析器和编译器将会忽略它。  
CodeBehind属性在Visual Studio .NET（非VS2005）中用来指定包含的代码隐藏文件。

Import  Import指令仅包括一个Namespace属性。所指定名字空间被明确地导入应用程序中，使其所有的类和接口都可用。导入的名字空间可以是.NET Framework的一部分或者用户自定义的名字空间。

以下是一个典型的Import指令：  
< %@ Import Namespace="System.Data" %>

只能有一个Namespace属性。如果需要导入多个名字空间，那么需要使用多个Import指令。  
下列名字空间自动导入到所有的Web应用程序中，所以没必要使用Import指令。  
l System  
l System.Collections  
l System.Collections.Specialized  
l System.Configuration  
l System.IO  
l System.Text  
l System.Text.RegularExpressions  
l System.Web  
l System.Web.Caching  
l System.Web.Security  
l System.Web.SessionState  
l System.Web.UI  
l System.Web.UI.HtmlControls  
l System.Web.UI.WebControls

Assembly  Assembly指令用于在编译过程中将一个程序集链接到当前应用程序。这样可以使所有程序集的类与接口对应用程序都是可用的。  
提示：典型的程序集是.dll或.exe文件，这将在下一章中详细讲解。

由于在编译时引用程序集，所以可以使用Assembly指令绑定程序集，然后在运行时将其加载到应用程序池中。

位于应用程序集缓存（也就是位于bin目录和App\_Code目录中的代码文件）中的程序集可自动连接到应用程序。因此，任何位于bin目录的程序集，或者由App\_Code目录中的代码编译而来的任何程序集，都不需要使用Assembly指令实现连接。

Assembly指令包括两个属性：Name和Src。Name属性是一个字符串，表示连接到应用程序的程序集名字，它不能包含路径。Src属性则是指向源文件的路径（只能为相对路径），这些文件将被动态编译和连接。

每个程序集指令只能有一个属性。如果需要连接多个程序集，则应使用多个Assembly指令。  
Assembly指令类似于：  
< %@ Assembly %>  
< %@ Assembly Src="sources/SomeSourceFile.cs" %>

脚本块  
典型的global.asax文件中包含大量代码，这些代码包含在以script标签起止的脚本块中：  
如果使用代码隐藏，虽然代码隐藏文件中的代码本身没有附加script标签，但包含在代码隐藏文件中的代码与脚本块中的代码是等效的。  
脚本块中的代码可以包含事件处理程序或者方法，下文将对此进行讲解。

事件  
如同Web页和控件可以公开事件一样，应用程序中的Application对象和 Session对象也能够公开事件。这些事件能被global.asax文件或指定的文件中的事件处理程序处理。

例如，当应用程序开始执行时(如web启动时)，触发 Application\_Start事件；

当应用程序结束时，触发Application\_End事件。

Application的某些事件是每当页面请求时触发(如：Application\_BeginRequest)，而其他一些事件，例如Application\_Error，则仅在特定情况下触发。

示例18-2中的global.asax文件代码说明了Application\_Start 和Application\_End事件。

示例18-2中的Application\_Start事件设置了两个Application属性：一个是名为 strConnectionString的字符串，一个是名为arBooks的字符串数组。事件处理程序方法调用

一个名为WriteFile的辅助方法，它包含在global.asax文件中。该辅助方法将一个字符串写入日志文件中（一般也是在应用启动的时候记录日志）。以下是示例18-2中的WriteFile方法代码：  
void WriteFile(string strText)  
{  
System.IO.StreamWriter writer =  
new System.IO.StreamWriter(@"C:\test.txt",true);  
string str;  
str = DateTime.Now.ToString(  ) + "  " + strText;  
writer.WriteLine(str);  
writer.Close(  );  
}

WriteFile 是一个简单的记录日志的方法。该方法初始化一个基于文本文件的 StreamWriter对象，并对c:\test.txt进行硬编码。它在文件中添加了一个时间戳，并写入通过方法传递的字符串。 StreamWriter方法的布尔值参数为true，其表示如果文件已经存在，那么将文本行追加到文件中。如果文件不存在，则创建一个文件。  
Application\_End事件处理方法调用了另一个WriteFile方法，它添加了一个日志条目以记录应用程序结束。

为了查看这两个事件处理程序的结果，可对global.asax进行一些无意义的编辑，并保存文件。此时将强制结束应用程序。然后请求虚拟目录中的任意URL地址。例如，使用上一章中的一个网页——实际上无论哪一个——或者一个自己创建的网页。示例18-3显示了日志文件内容。  
示例18-3：Test.txt摘录  
8/26/2006 5:46:23 PM  Application Starting  
8/26/2006 6:13:35 PM  Application Ending  
8/27/2006 10:17:39 PM  Application Starting  
8/27/2006 10:18:23 PM  Application Ending  
8/27/2006 10:18:36 PM  Application Starting  
如同Application对象的Start和End事件一样，Session对象也拥有Session\_ Start和Session\_End事件。这将允许应用程序每次启动和结束过程中为每个会话都运行代码。  
如示例18-4中高亮显示的方法名所示，其包括了global.asax文件中所有可能的应用程序事件处理程序。在页面请求被接受、处理和呈现过程中，可以容易地查看应用程序生命周期。  
示例18-4：Global.asax事件说明  
< %@ Application  Language="C#" %>  
以下是页面请求触发的所有事件，以触发顺序排序：  
Application\_BeginRequest  
当ASP.NET开始处理每个请求时触发。在这个事件处理中的代码将在页面或者服务处理请求之前执行。  
Application\_AuthenticateRequest  
在验证请求之前触发。（正如第12章介绍的，验证是确认用户就是他所说的那个人的过程）在这个事件处理程序的代码中允许实现自定义安全管道。  
Application\_AuthorizeRequest  
在为请求授权之前触发。（授权是确定是否请求用户具有访问资源的权限的过程，在  
第12章已经介绍过）在这个事件处理程序的代码中允许实现自定义安全管道。  
Application\_ResolveRequestCache  
在ASP.NET确定是否应该生成新的输出，或者由缓存填充前触发。无论何种情况，都将执行该事件处理程序中的代码。  
Application\_AcquireRequestState  
在获取会话状态之前执行。  
Application\_PreRequestHandlerExecute  
在将请求发送到服务于请求的处理程序对象之前触发。当事件触发后，页面将由HTTP处理程序处理请求。  
Application\_PostRequestHandlerExecute  
当HTTP处理程序与页面请求一起完成时触发。此时，Response对象将获得由客户端返回的数据。  
Application\_ReleaseRequestState  
当释放和更新试图状态时触发。  
Application\_UpdateRequestCache  
如果输出被缓存，那么缓存更新时将触发。  
Application\_EndRequest  
当请求结束时执行。  
Application\_PreSendRequestHeaders  
在向客户端发送HTTP头之前触发。如果启用响应缓存，这意味着直到所有数据都准备好（默认条件），都不会发送任何数据。该事件总是在Application\_EndRequest事件之后。如果禁用响应缓存，那么无论何时将数据发送给客户端，都将触发该事件。响应控制由Page指令的一个属性，或者Web服务的WebMethod属性控制。  
Application\_PreSendRequestContent  
向客户端发送HTTP内容之前触发。和Application\_PreSendRequestHeaders事件一样，Application\_PreSendRequestContent事件能否被触发取决于响应缓存是否可用。  
以下列举应用程序事件，它们在特定条件下触发：  
Application\_Start  
当应用程序启动时触发。当首次请求应用程序虚拟目录中的任何页面时，将启动应用程序，同时如果应用程序已经运行，则不触发该事件。  
Application\_End  
应用程序结束时触发。无论何时修改了配置文件（global.asax、global.asax.cs、  
global.asax.vb或者web.config），或者服务器崩溃或者重启，应用程序都将结束。通常在该事件处理程序中执行清除功能的代码，例如关闭[数据库](http://lib.csdn.net/base/14" \t "_blank" \o "MySQL知识库)连接。  
Session\_Start  
每个会话开始时触发，这是放置具体会话代码的地方。  
Session\_End  
会话结束时触发。它为保存存储在会话中的任何数据提供了机会。  
Application\_Disposed  
当CLR从内存中移除应用程序时触发。  
Application\_Error  
无论在应用程序中何时何处发生未处理的错误都将触发。它提供了一个实现通用应用程序错误处理的好机会。  
使用try...catch语句块能够在代码中处理特定错误，也可以使用Page指令的ErrorPage属性来捕获页面级错误。使用这些方式处理任何错误都不会触发Application\_Error事件。  
为了测试新版的global.asax，在示例18-5中创建一个网页。查看GlobalEvents站点。当该网页运行时，将看到如图18-4所示的页面。  
示例18-5：GlobalEvents网站的default.aspx  
< %@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"  
Inherits="\_Default" %>  
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">  
Global Events  
Text="End Session"  
OnClick="btnEndSession\_Click" />  
Text="Generate Error"  
OnClick="btnError\_Click" />  
 图18-4：GlobalEvents  
如图18-4所示，可看到触发了一系列应用程序事件。在这些事件执行的中途，呈现了.aspx页面自身，接着是另一些应用程序事件。  
警告：示例GlobalEvents必须在一个真实的IIS虚拟路径下才能良好运行。  
页面第一次显示时触发Session\_Start事件，而在随后的显示中则不再触发该事件。 这是因为请求是相同的会话。单击“End Session”按钮，调用Session.Abandon方法，这将结束当前会话。下一次页面被提交到服务器时，将再次触发Session\_Start 事件。  
Post按钮提供了一个简单的再次提交页面的方法。  
示例18-4中大部分的应用程序事件处理程序都使用了Response.Write方法，以便显示触发的事件。然而，Application\_Start和Application\_End方法调用了WriteFile方法。如果在这些事件处理程序中试图使用Response.Write方法，那么在页面中将不会  
显示。因为那时用于呈现的页面会话还没有运行。但是，当检查日志文件c:\test.txt时，将看到应用程序启动和结束所显示的条目。  
示例18-4所示的global.asax文件说明了一种使用Application\_Error事件的方法。以下列举了这些代码：  
protected void Application\_Error(Object sender, EventArgs e)  
{  
string strError;  
strError = Server.GetLastError(  ).ToString(  );  
if (Context!= null)  
  Context.ClearError(  );  
Response.Write("Application\_Error" + "  
");  
Response.Write("Error Msg: " + strError + "  
" +  
  "End Error Msg  
");  
}  
该事件处理程序使用HttpServerUtility对象的GetLastError方法，以报告最后出现的错误。这个错误被转换为一个字符串，并被指定给一个字符串变量：  
strError = Server.GetLastError(  ).ToString(  )  
接下来调用HttpContext对象的ClearError方法来清除目前HTTP请求中的所有错误：  
Context.ClearError(  )  
如果没有将错误清除，那么错误将显示在客户端浏览器，并且还是无法看见Respons- e.Write方法的显示结果。  
最后，Response.Write方法显示一个信息，当前的错误将显示在客户端。  
另一种向用户报告错误的方式是显示自定义错误处理页面。为此，需要使用以下代码行替代Application\_Error事件处理程序中的Response.Write方法：  
Response.Redirect("CustomErrorPage.aspx?Msg=" +  
  Server.UrlEncode(strError));  
以上代码行调用HttpServerUtility对象的UrlEncode方法，其将错误信息作为一个QueryString参数传递给CustomErrorPage.aspx页面中的自定义错误处理代码。 CustomErrorPage.aspx页面有一个名为lblMessage的Label控件。以下是页面中的Page\_Load方法代码：  
void Page\_Load(Object Source, EventArgs E)  
{ lblMessage.Text = Request.QueryString["Msg"]; }  
Default.aspx中的Generate Error按钮故意触发一个错误，以便查看错误处理。该按钮的单击事件处理程序的代码如下所示，它将除零异常：  
protected void btnError\_Click(object sender, EventArgs e)  
{  
int a = 5;  
int b = 0;  
int c;  
c = a / b;  
}  
服务器端包括  
使用服务器端包括能够实现在应用程序中包含外部源代码文件。在编译之前，包括文件中的代码将被添加到global.asax文件中。尽管应用程序的语言可能与包括文件的语言不同，但是用于包括文件的语言必须与global.asax文件所使用的语言匹配。  
下面是用于服务器端的语法：  
在这个句法中，PathType类型可以是表18-1所示之一。  
表18-1  PathType属性

|  |  |
| --- | --- |
| 路径类型 | 说  明 |
| File | 文件名是包含global.asax文件的目录的相对路径 |
| Virtual | 文件名是包含网站虚拟目录的虚拟路径 |

查看示例18-4所示的global.asax文件，将下面的代码添加到第二行中：  
  
创建一个新的名为IncludeFile.cs的文本文件，将该文件和global.asax存储在同一目录下。这个文件需要与global.asax文件一样的一对脚本标签。  
  
将global.asax页面中的WriteFile方法复制到包括文件中，然后注释（或者删除）global.asax页面中的WriteFile方法。这样包括文件应类似于示例18-6。  
  
示例18-6：具有包括文件的global.asax  
  
如果运行任意一个网页，那么将不会与先前有什么区别，因为您所做的只是把一个文件中的代码转移到另外一个文件中。  
  
如果CLR监视global.asax文件的变化，并和重新启动应用程序一样，CLR也监视包括文件的变化。如果包括文件发生变化，那么应用程序也会重新启动。  
  
对于在多个应用程序中所包括的相同标准代码而言，包括文件非常有用。这些通用代码可能包括数据库访问方法、写入日志记录、错误处理管道、登录或者每个应用程序的基础类型代码片段。  
  
对象声明  
  
在global.asax文件中包括代码的另一个方法是是声明object标签。这些声明的静态对象要么是Application，或者是Session对象。这样就可以在应用程序或者每个会话过程中使用。