ASP Microsoft .net

Etape par étape

4ème partie : Utilisation du Global.asax

(Révision : 1 du 11/04/2005 – 14 pages)

Avertissement :

Ce document peut comporter des erreurs. Cependant, tout a été mis en œuvre afin de ne pas en inclure dans ce texte. Tout code qui trouve sa place ici a été testé au préalable.

Tables des matières :

Bibliographie

Site MSDN (http://msdn.microsoft.com)

Travail de fin d'étude : « Etude détaillée et application des Services Web .NET », Genon Nicolas et Dewit Stéphane,Haute Ecole Rennequin Sualem.

Developing Web Applications with Microsoft Visual Basic.NET and Visual C#.NET (MCAD/MCSD Self-Paced Training Kit), Jeff Webb and Microsoft Corporation.

Introduction au Global.asax

La classe Global est la classe capable de gérer des évènements du niveau application. Elle se trouve dans un fichier appellé Global.asax (plus précisément Global.asax.cs ou Global.asax.vb mais nous verrons cela plus loin dans ce document). Le Global.asax est optionnel mais il peut faciliter le développement ainsi que la maintenance d'applications.

On l'utilisera, par exemple , pour écrire une entrée dans un fichier de logs lorsqu'une exception est lancée et n'est pas gérée. Nous verrons également d'autres utilisations possibles sans toutefois les citer toutes tant elles sont nombreuses.

Concrètement, la classe Global est une classe dont il n'existe qu'une instance (ce que l'on appelle « singleton » dans le jargon).

La classe Global est compilée lors du premier appel (comme toutes les classes en asp.NET d'ailleurs) pour en faire une classe MSIL.

Fonctionnement d'une application asp.NET

Le runtime d'asp.NET maintient un pool d'objets HttpApplication. Ainsi, à chaque requête, le runtime prend un de ces objets et l'attache à la requête.

Cet objet HttpApplication ne peut être attaché à une autre requête tant que celle-ci est en cours de traitement.

Lorsque le traitement est terminé, l'objet est remis dans le pool en vue d'une nouvelle utilisation.

Présentation générale du fichier Global.asax

Ce fichier se trouve dans le répertoire racine de l'application asp.NET.

Il est configuré pour rejeter automatiquement toute demande directe par l'url. On ne peut, donc, ni télécharger ni voir le code contenu dans la classe Global.

La classe Global contient plusieurs méthodes qui ont la forme Level_EventName(object sender, AppropriateEvent e).

Voici un exemple de fichiers créés par Visual Studio.NET 2003 :

Global.asax

```
<%@ Application Codebehind="Global.asax.cs" Inherits="Developpez.Global" %>
```

Ce fichier fait référence au Global.asax.cs.

Global.asax.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Web;
using System.Web.SessionState;

namespace Developpez
{
	/// <summary>
	/// Description résumée de Global.
	/// </summary>
	public class Global : System.Web.HttpApplication
	{
		private System.ComponentModel.IContainer components = null;

		public Global() { InitializeComponent(); }

		protected void Application_Start(Object sender, EventArgs e) { }

		protected void Session_Start(Object sender, EventArgs e) { }

		protected void Application_BeginRequest(Object sender, EventArgs e) {}

		protected void Application_EndRequest(Object sender, EventArgs e) { }

		protected void Application_AuthenticateRequest(Object sender,
EventArgs e) { }

		protected void Application_Error(Object sender, EventArgs e) { }

		protected void Session_End(Object sender, EventArgs e) { }

		protected void Application_End(Object sender, EventArgs e) { }

		#region Code généré par le Concepteur Web Form
		/// <summary>
		/// Méthode requise pour la prise en charge du concepteur - ne modifiez
pas
		/// le contenu de cette méthode avec l'éditeur de code.
		/// </summary>
```

```
            private void InitializeComponent()
            {
                this.components = new System.ComponentModel.Container();
            }
            #endregion
        }
}
```

Utilisation d'un objet Application

Comme cela a déjà expliqué dans le tome 3, une variable d'application est une variable partagée par tous les utilisateurs contrairement aux variables de session qui ne sont accessibles que par un seul utilisateur.

Reprenons donc l'exemple du tome 3 page 9 :

```
if (Application["NbUsers"]!=null)
  Application["NbUsers"]= Convert.ToInt32(Application["NbUsers"])+1;
else
  Application["NbUsers"] = 1;
```

Quels sont les désavantages de cette syntaxe ?

On pourrait écrire ***Application["NombreUsers"]*** ou plus simplement, suite à une faute de frappe, ***Application["NUsers"]*** , ce qui passera sans même un avertissement à la compilation. Malheureusement, le résultat obtenu risque de différer de celui escompté.

Ainsi ,

```
if (Application["NbUsers"]!=null)
  return Application["NUsers"] ;
```

pourrait renvoyer une exception car ***Application["NUsers"]*** peut être ***null*** alors que l'on pensait avoir gérer ce cas.

Il en va de même pour les conversions nécessaires. Imaginons que l'on mette une classe quelconque dans une variable d'application. Si l'on applique un casting sur cet objet alors qu'il n'est pas possible de faire ce casting sur ce type d'objet, une exception est levée.

Utilisation de variables statiques

Pour remédier à ces problèmes, il est possible de définir ce type de variable au sein même de la classe Global.

```
public class Global : System.Web.HttpApplication
{
public static readonly string ConnectionString="connection infos" ;
public static int numeroBidon;

// Méthodes
}
```

Dans ce cas, l'appel se fait par

```
string strConn = Global.ConnectionString ;
```

Ce qui évite les problèmes de nommage ainsi que de casting tels qu'ils pourraient survenir avec la technique précédente.

Il est nécessaire que la variable soit statique afin d'être accessible à partir des autres classes de l'application. Dans le cas contraire, elle est seulement utilisable comme Request State (variable qui a comme durée de vie que le temps nécessaire à l'exécution de la requête), c'est à dire dans Application_BeginRequest(object sender, EventArgs e) et Application_EndRequest(object sender, EventArgs e). En dehors de ces fonctions, la variable n'est pas encore ou n'est plus définie.

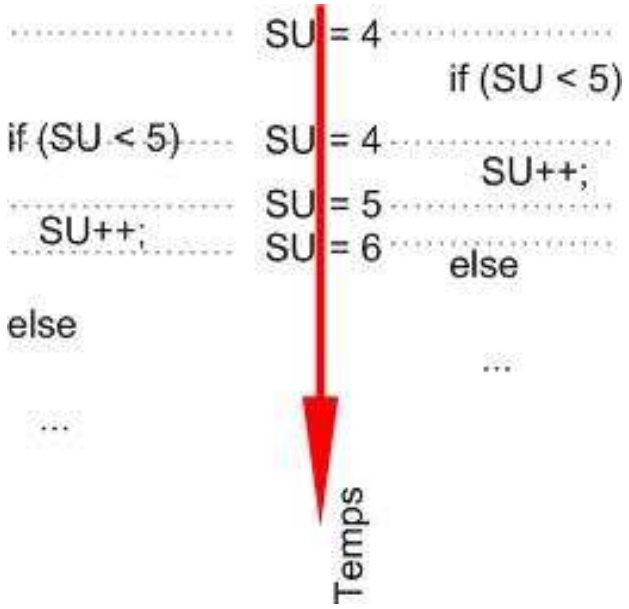
Lock des variables d'Application

Imaginons une application Web qui ne pourrait être utilisée que par cinq utilisateurs simultanément. Cela pourrait être traité à l'aide des évènements Application_Start, Session_Start et Session_End.

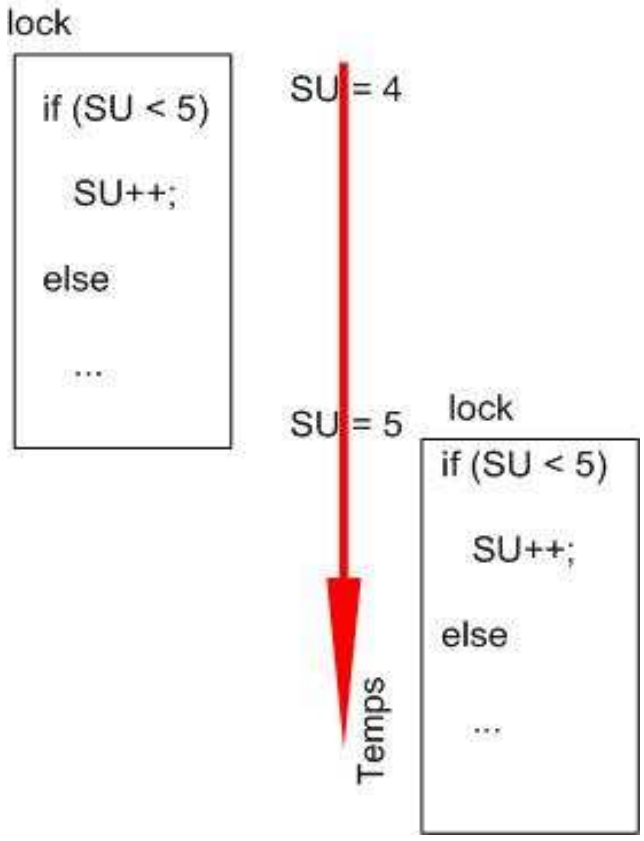
```
        /// </summary>
        public class Global : System.Web.HttpApplication
        {
                public static readonly int SimultanousUsers;

                protected void Session_Start(Object sender, EventArgs e)
                {
                        if (SimultanousUsers == 5)
                                Response.Redirect("./Sorry.html", false);
                        else
                                SimultanousUsers ++;
                 }

                protected void Session_End(Object sender, EventArgs e)
                {
                        SimultanousUsers --;
                 }

                protected void Application_Start(Object sender, EventArgs e)
                {
                        SimultanousUsers = 0;
                 }


}
```

Dans ce cas, nous effectuons un traitement, il est donc conseillé d'utiliser les locks. Ceux-ci permettent de verrouiller des parties de code. Il n'est ainsi pas possible d'exécuter simultanément deux parties de code se trouvant dans un bloc verrouillé.

Revenons à notre exemple. Sans les locks, nous pourrions arriver à la situation suivante (nombre d'utilisateurs égal à 6) :



Cela ne peut arriver avec les locks :



Il ne reste plus qu'à voir comment verrouiller ce code :

```
/// </summary>
public class Global : System.Web.HttpApplication
{
        public static readonly int SimultanousUsers;

        protected void Session_Start(Object sender, EventArgs e)
        {
                lock(this)
                {
                        if (SimultanousUsers == 5)
                                Response.Redirect("./Sorry.html", false);
                        else
                                SimultanousUsers ++;
                }
         }
```

```
        protected void Session_End(Object sender, EventArgs e)
        {
                SimultanousUsers --;
         }

        protected void Application_Start(Object sender, EventArgs e)
        {
                SimultanousUsers = 0;
         }


}
```

Description des méthodes principales

Application_Start

Se déclenche lorsque l'application démarre. Cette méthode ne sera de nouveau appelée que lors du prochain redémarrage de l'application.

C'est généralement dans cette fonction que l'on initialise l'application (Il s'agit d'une habitude prise par les développeurs).

Session_Start

Se déclenche lorsqu'un nouveau client se connecte à l'application.

Session_Stop

Se déclenche lorsque le client en cours se déconnecte (plus exactement lorsque la session se termine, soit 20 minutes, qui est la valeur par défaut après la dernière action du client) ou que la client a explicitement demandé un abandon de session par la commande Session.Abandon() ;

Application_BeginRequest

Se déclenche lors de chaque requête.

Application_EndRequest

Se déclenche à la fin de chaque requête.

Application_Error

Se déclenche lorsqu'une exception n'est pas gérée.

Application_End

Se déclenche lorsque l'application se termine (lorsque plus aucun client n'a de session en cours ou que l'on demande un shutdown du serveur). Cette méthode n'est donc appelée qu'une fois dans le cycle de vie de l'application.

Description des autres méthodes

Application_Init

Cette méthode se déclenche avant l'Application_Start. Elle peut être utilisée pour initialiser l'application.

Application_Dispose

Se déclenche lorsque l'application a terminé toutes les requêtes. C'est ici qu'il faut utiliser le Garbage Collector afin de nettoyer la mémoire des données restantes et non utilisées.

Application_PreRequestHandlerExecute

Se déclenche juste avant que asp.NET ne commence à exécuter une page ou un web service. A partir de ce point, le Session State est accessible.

Application_PostRequestHandlerExecute

Se déclenche quand le handler d'asp.NET finit son exécution.

Application_PreSendRequestHeaders

Se déclenche juste avant que asp.NET n'envoie l'en-tête http au client. Les informations de l'en-tête peuvent être modifiées en utilisant cet événement.

Application_PreSendRequestContent

Se déclenche juste avant que asp.NET n'envoie le contenu vers le client.

Application_AuthenticateRequest

Se déclenche lorsque l'utilisateur accède à une ressource pour laquelle il faut être authentifié dans le cas où cette authentification se fait par le biais des mécanismes mis en place par asp.NET.

Application_AuthorizeRequest

Se déclenche lorsqu'un utilisateur accède à une ressource pour laquelle il doit être autorisé.

Exemples d'utilisation

Calcul du temps nécessaire à l'exécution d'une requête

```
...

namespace Developpez
{
	public class Global : System.Web.HttpApplication
	{
		static public int temps;
		private DateTime debut;

		protected void Application_BeginRequest(Object sender, EventArgs e)
		{
			debut = DateTime.Now;
		}

		protected void Application_EndRequest(Object sender, EventArgs e)
		{
			temps = debut – DateTime.Now;
		}

	...
	}
}
```

Dans une page, il est possible d'afficher le temps nécessaire à l'exécution de la requête par

```
LaTime.Text = Global.temps.ToString() ;
```

Compteur de visites simultanées

```
...
```

```
namespace Developpez
{
	public class Global : System.Web.HttpApplication
	{
		public static int nbUsers;

		protected void Application_Start(Object sender, EventArgs e)
		{
			nbUsers = 0;
		 }

		protected void Session_Start(Object sender, EventArgs e)
		{
			nbUsers++ ;
		 }

		protected void Session_End(Object sender, EventArgs e)
		{
			nbUsers-- ;
		 }
	...
	}
}
```

Compteur de hits

```
namespace Developpez
{
	public class Global : System.Web.HttpApplication
	{
		static public int nbHits;

		protected void Application_Start(Object sender, EventArgs e)
		{
			 nbHits = 0;
		}

		protected void Application_BeginRequest(Object sender, EventArgs e)
		{
			nbHits++;
		}

		...
	}
}
```

Renvoi vers une page d'erreur

```
...

namespace Developpez
{
	public class Global : System.Web.HttpApplication
	{
		...

		protected void Application_Error(Object sender, EventArgs e)
		{
			string exMessage =
Server.GetLastError().InnerException.Message ;
			// … Utilisation du message d'erreur (enregistrement de log, mise
dans une variable de session pour l'affichage, …)
			Response.Redirect("Erreur.htm") ;
		}

	...
	}
}
```

Conclusion

Un conseil. N'utilisez le Global.asax que si vous souhaitez régulièrement travailler avec car vous serez très vite accro à celui-ci.

Plus sérieusement, au travers de ce document, on se rend compte qu'il est très utile et très simple d'utilisation. Il permet dans certains cas d'obtenir de bonnes statistiques, dans d'autres cas, de faciliter l'écriture du code ou encore simplement d'éviter d'avoir d'affreux messages d'erreur à l'affichage.