ASP Microsoft .net

Etape par étape

1ère partie : Introduction à asp.NET

(Révision : 2 du 05/11/2004 – 13 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

Gérard Leblanc, c# et .NET, ed. Eyrolles

MSDN

Copyright © 2004 Danse Didier. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300000€ de dommages et intérêts.

Au travers de cette série de documents, nous allons apprendre à créer une page en ASP (Active Server Page) basé sur la technologie .NET. Mais qu'est ce donc que .NET ? Nous sommes à peine à la fin de la deuxième ligne et déjà quelques personnes se sentent déroutées. C'est pourquoi, nous allons essayer d'être le plus progressif possible.

Pré requis

Le langage utilisé dans ce document pour le code des traitements sera C#. Il existe d'autres langages tels que VB.NET et delphi.NET. Nous n'en parlerons que très peu ici.

Apprendre ASP.NET est ici notre but principal, c'est pourquoi, il est préférable d'avoir des bases de C# (langage choisi dans le cadre de ces tutoriaux) et d'html principalement (qui est nécessaire à la création de pages web).

Rm di Scala parle très bien du c# dans son document Le langage c#, les premiers pas. Il existe également un autre document sur les bases du c#, comprenant des exercices, écrit de nouveau par Rm di Scala. Si une information manque à vos yeux dans ces documents , peut être se trouve-t-elle dans la Foire Aux Questions .NET.

Pour ce qui est de l'apprentissage de l'html, je vous renvoie vers http://cyberzoide.developpez.com/html/.

Il est également avantageux de connaître le JavaScript mais cela n'est pas indispensable. Nous n'en parlerons pas donc dans ces documents.

De plus il est intéressant d'avoir regarder les différents menus de l'utilitaire que nous allons utiliser pour développer nos pages web. Il s'agit de WebMatrix. En voici une petite présentation…

WebMatrix

D'apparence et d'utilisation forts semblables à Visual Studio .NET, Webmatrix est un utilitaire développé dans le but de permettre à chacun de créer son site web en .NET.

Pourquoi utiliser cet outil qui ne permet la création que de site web alors qu'il existe VS.NET ? Pour la simple raison que WebMatrix est gratuit ! Ainsi, tout le monde peut développer son site web sans devoir posséder un outil de développement aussi puissant. Pour le télécharger, rien de plus simple : il suffit de cliquer sur le lien suivant : WebMatrix.

Par ailleurs, un autre gros avantage apparaît si vous utilisez Windows Xp Home. Effectivement, Internet Information Service, le serveur web inclus dans Microsoft Xp Pro, n'est pas installé sur ce type de système d'exploitation. WebMatrix permet d'exécuter un serveur afin de voir les pages asp.NET.

Effectivement, il possède son propre serveur pour asp.NET qui permet d'exécuter ses pages de manière locale. Ainsi, il est possible à tous de tester ses pages sans même une connexion internet.

De plus, Visual Studio .NET travaille sous forme de projets, ce qui est assez déroutant pour le débutant car toute une série de fichiers sont créés. Ce n'est pas le cas avec WebMatrix. A une page créée, un seul fichier est associé.

Après avoir présenter WebMatrix en quelques mots, nous allons pouvoir réellement entrer dans le vif du sujet et voir comment créer notre page.

Présentation de asp.NET

Asp.NET est basé sur la technologie .NET. Il permet la programmation d'applications Web dynamiques, du côté du serveur. Les navigateurs Web, à l'aide de pages au format html, servent donc d'interface entre l'application .NET et l'utilisateur.

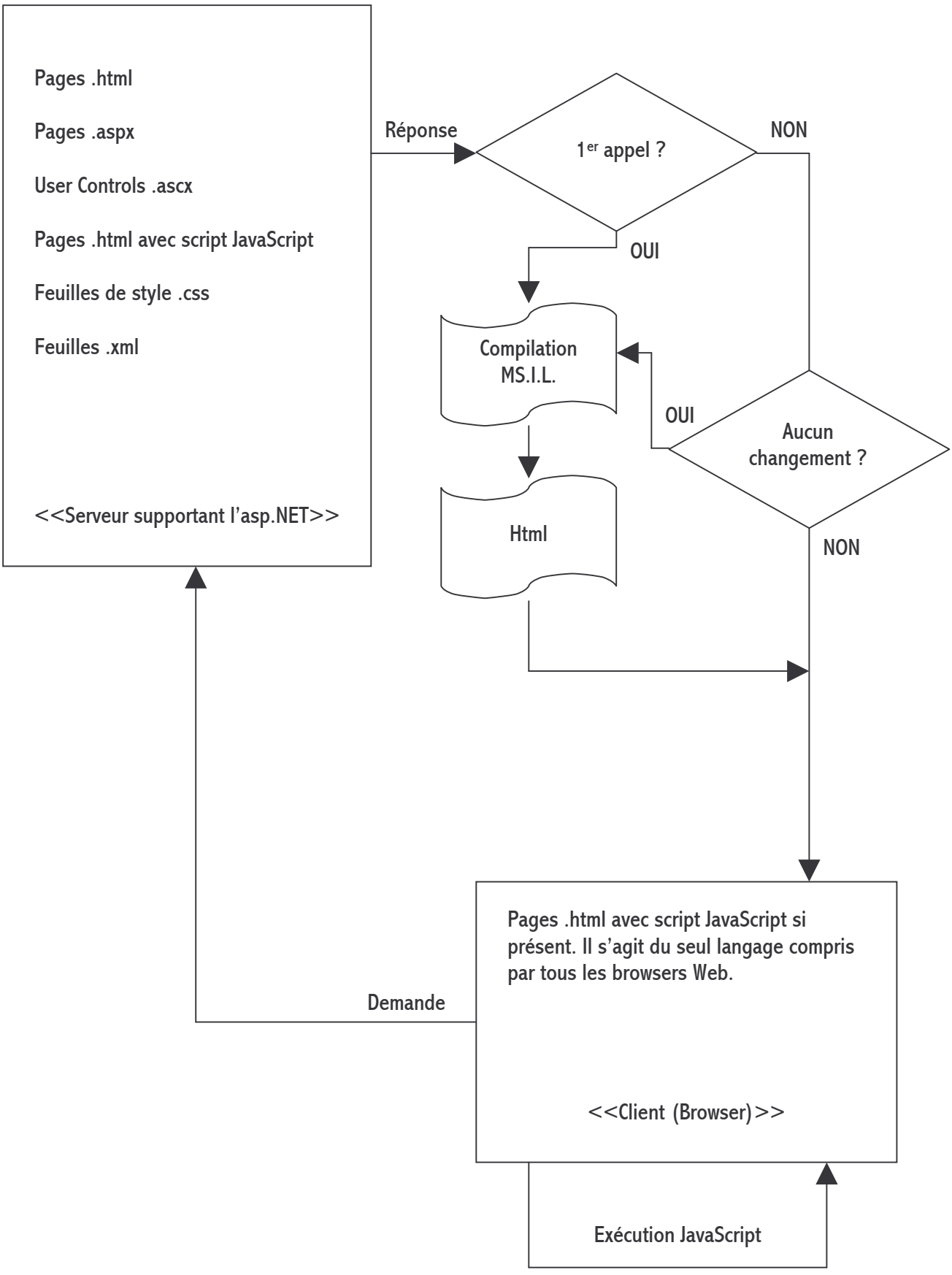
Contrairement à asp, où le code était inclus directement dans la partie html, asp.NET est un langage compilé. Pour plus d'explications, je vous renvoie au Mémoire sur .NET.

La partie html (interface) et la partie c# (traitements) peuvent ainsi être séparées au sein d'un même fichier ou même dans des fichiers différents. Il est ainsi possible, pour un designer web, de mettre en forme une page web sans pour cela connaître le langage c#. L'inverse est également vrai.

Ainsi, il est possible de travailler de manière séparée et d'avoir des personnes spécialisées pour chaque partie de la page. Il suffit de connaître l'interface entre le code et ce qui est affiché, c'est à dire le nom des WebForms (nous verrons ce qu'est un webform un peu plus loin dans ce tutorial). Ceci est un gros avantage d'asp.NET.

Il est important de comprendre le fonctionnement de l'architecture client/serveur pour développer dans cette architecture. C'est la raison pour laquelle nous allons voir cette architecture de manière schématisée.

Architecture Client/Serveur



Lors d'une demande venant d'un navigateur arrive au serveur, ce dernier vérifie si la page a déjà été compilée. Si c'est le cas, la page, déjà au format html est envoyée au navigateur du client. Dans le cas contraire, la page est la page au format asp.NET est d'abord compilée au format M.S.I.L., MicroSoft Intermediate Language, langage qui est généré lors de tout appel à des fichiers .NET (de nouveau, le Mémoire sur .NET permet de répondre à certaines questions), ensuite, la page est générée au format html.

Cette page peut contenir des scripts utilisant des langages utilisés du côté du client. La particularité de tels scripts est de ne pas devoir effectuer de retours serveur et ainsi limiter les échanges. Cependant nous n'en parlerons pas ici.

Nous allons désormais entrer dans le vif du sujet…

Structure générale d'une page asp.NET

Une page asp.NET possède une extension ***.aspx***. Nous verrons comment il est possible de séparer le code de la partie interface dans un prochain tome (rubrique Code-Behind).

Cette page a la forme générale suivante :

```
<head>
<script language= "c#" runat="server">
        type  fct (type i) { … }
</script>
<script language= "javascript">
        function f ;
</script>
</head>
<body>
        <table>
                …
        </table>
        <asp :Label [Propriétés]></asp :Label>
        <% =fct(5) %>
</body>
```

Plusieurs points peuvent ressortir de cette « page-type ». On peut les séparer en trois parties.

La première partie fait apparaître les scripts. Effectivement, on remarque que du c# et du javascript cohabitent dans la même page. En quelques lignes, nous verrons ce qu'il est possible de mettre dans les balises de ce type.

La deuxième, permet de montrer que le code html est identique au code html d'une page web statique.

Et la dernière montre les balises qui déroutent le plus le non initié, c'est à dire ***<asp :…> et <% … %>***

Il s'agit en fait de, respectivement, la balise d'ouverture pour un WebForm asp.NET et d'une balise de fonction. Nous allons les détailler quelque peu.

Script de traitements (c#)

Tout script écrit en c# pour asp.NET doit se trouver entre les balises suivantes

```
<script language= "c# " runat= "server ">
        …
</script>
```

Le tout doit se trouver entre la balise html ouvrante ***<head>*** et celle fermante ***</head>.***

Si l'on avait utilisé un autre langage tel que VB.NET, on aurait eu

```
<script language= "VB.NET " runat= "server ">
        …
</script>
```

Nous allons maintenant détailler ce qui se trouve dans ces balises.

La propriété ***language*** permet de déclarer le langage dans lequel le script est écrit. Dans le premier cas, et ce sera d'ailleurs comme cela au sein de ces tutoriaux, il s'agit de ***c#***. Par page .aspx, on ne peut définir qu'un seul langage .NET pour le script.

Il n'est possible d'avoir dans une page asp.NET qu'un seul langage utilisant la technologie .NET, ainsi

```
<script language= "c# " runat= "server ">
        …
</script>
<script language= "VB.NET " runat= "server ">
        …
</script>
```

n'est pas valide.

La valeur ***"server"*** assignée à la propriété ***runat*** signale que le script doit s'exécuter sur le serveur d'application. Le navigateur client ne recevra que du code html.

Entre les balises ***<body>*** et ***</body>,*** nous allons trouver les balises suivantes : ***<form runat= "server ">*** et ***</form>*** qui signifient que nous utilisons un formulaire qui doit être traité par le serveur. Effectivement, le client ne doit recevoir que du code html, le seul code compris par tous les navigateurs. Les balises <asp : seront donc traitées avant l'envoi de la page html.

Syntaxe d'un WebForm asp.NET

Tout d'abord, nous allons répondre à la question que l'on se pose en voyant le titre: « qu'est ce qu'un WebForm ? ». Il s'agit d'un contrôle qui se trouve du côté serveur et qui est donc manipulé par les scripts de traitement. Il n'est pas visible du côté du client puisque la page que celui-ci reçoit ne contient que du html.

Pour insérer un WebForm asp.NET, nous utiliserons soit ***<asp : … />*** , soit
<asp :Type [Propriétés]>
…
</asp :Type>
suivant la nature de celui-ci. Nous verrons dans le tome 2 lequel choisir et pourquoi il existe ces deux syntaxes.

Tous les webforms ont des propriétés qui sont identiques. Nous les verrons également dans le tome 2. Ensuite, nous verrons dans le détail chacun des webforms dits simples.

Exécution et appel de fonctions dans une page Web

Il est possible d'appeler du code directement dans une page web. Ainsi on peut trouver au sein d'une page web ceci

```
<body>
<form runat="server ">
        …
        <h2> Date du jour : <% =DateTime.Now.ToString() %> </h2>
        …
</form>
</body>
```

Ce qui donnera à l'affichage:

Date du jour : 29/02/2004 22:57:53

On peut également appeler des fonctions personnelles par cette méthode.

Dans ce cas, il suffit, dans la balise <% … %> de mettre =Nom_Fonction().

Voici un exemple qui y correspond :

```
<head>
<script language= "c#" runat= "server">
        string Demain()
        {
                DateTime Jour=DateTime.Now ;
                Jour=Jour.AddDays(1) ;
                return Jour.ToString() ;
        }
</script>
</head>
<body>
<form runat="server">
…
Date du jour : <% =DateTime.Now.ToString() %><br>
Date de demain : <% =Demain() %>
…
</form>
</body>
```

Qui donne:

… Date du jour : 29/02/2004 23:07:01
Date de demain : 1/03/2004 23:07:01 …

Le résultat sera le même si l'on exécute du code directement dans le code html. Cependant il est avantageux de séparer html et c# afin de clarifier le code de notre page.

Pour faire cela, il suffit de remplacer la fonction par le code proprement dit.

```
<head>
</head>
<body>
<form runat="server">
        …
Date du jour : <% =DateTime.Now.ToString() %><br>
Date de demain :
<%
DateTime Jour=DateTime.Now ;
Jour=Jour.AddDays(1) ;
Response.Write(Jour.ToString()) ;
 %>
…
</form>
</body>
```

Quelques mots sur le tome 2

Pour générer des pages qui contiennent des informations qui peuvent varier, comme c'est le cas pour des sites d'achats ou encore des forums, on utilise la programmation côté serveur à l'aide de scripts tels que ceux dont on vient de parler. On peut évidemment trouver d'autres utilisations à la programmation web dynamique. Dans les tomes suivants, il s'agira de gérer l'inscription d'un utilisateur, de modifier ses données et d'effectuer d'autres opérations le concernant.

Au travers du tome 2, nous allons progressivement créer une page d'ouverture d'un compte pour un site quelconque.

Mais avant de passer au tome 2, je tiens à remercier David Pédehourcq qui m'a soutenu et aidé à l'élaboration de ce premier tutorial.