Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

Gestion du cache dans les applications ASP .NET

Version 1.0

James RAVAILLE
http://blogs.dotnet-france.com/jamesr

Sommaire

1 Introduction

1.1 Présentation

Une bonne gestion de cache dans les applications ASP .NET, permet d'améliorer nettement les performances d'exécution des pages Web sur le serveur. Mais toute donnée mise en cache est une copie d'une autre donnée. Alors des effets de bord peuvent apparaître, comme les problèmes liés à la fraicheur des données.

Dans ce cours, nous vous proposons alors d'étudier les différentes fonctionnalités de gestion du cache, que propose la technologie ASP .NET :

- Mise en cache de données.
- Utilisation des dépendances de cache, vers un fichier et une table d'une base de données SQL Server.
- Mise en cache des pages ou parties de pages.

2 Utilisation du cache de données

2.1 Présentation du cache de données

Le cache de données est un contexte de données, situé côté serveur, qui est partagé entre tous les utilisateurs. Autrement dit, si un utilisateur ajoute ou modifie une donnée dans le contexte de cache, alors tout autre utilisateur peut lire cette donnée, et même la modifier. Il est possible d'ajouter tout type d'objet. Charge à l'utilisateur de convertir la donnée lue du cache, pour obtenir une donnée fortement typée.

Quelle est alors la différence entre le contexte d'application et le contexte de cache ? Si on s'en tient à la présentation du cache de données effectuée ci-dessus, on ne peut voir la différence. De nettes différences existent pourtant et concernant les données mises en cache :

- Leur durée de vie est limitée. Il est possible de spécifier deux durées de vie :
  - Une durée de vie *absolue* : durée de vie au bout de laquelle la variable de cache est automatiquement détruite. Le point d'origine temporel est définit lors de l'ajout de la donnée dans le cache.
  - Une durée de vie *glissante* : durée de vie au bout de laquelle la variable de cache est automatiquement détruite. Le point d'origine temporel est définit lors du dernier accès à la donnée dans le cache, part n'importe quel utilisateur.
- Il est possible de les lier à une « source de données » ou un « indicateur », pouvant être :
  - Une table d'une base de données. Le Framework .NET ne prend en charge que les bases de données SQL Server.
  - Un fichier texte ou binaire.
- Concernant la sécurité des threads, la classe *System.Web.Caching.Cache* (permettant de gérer le cache de l'application), est *thread-safe*. Autrement dit, les instances de cette classe ne peuvent être utilisées que par un seul thread à la fois. Pour sécuriser la gestion des données dans le contexte d'application, il est nécessaire d'encadrer les instructions de gestion des données, avec les instructions *Application.Lock()* et *Application.Unlock()*.

2.2 Manipulation des données en cache

2.2.1 L'objet Cache

L'objet intrinsèque *Cache* permet de gérer le cache de données dans les applications ASP .NET.

2.2.2 Ajout d'une donnée dans le contexte de cache

Il existe trois méthodes permettant d'ajouter une donnée dans le contexte de cache, créant ainsi des variables de cache :

- En appliquant la méthode *Add* sur l'objet *Cache*. Cette méthode retourne l'objet mis en cache. Toutefois, si cette variable de cache existe déjà, une exception est levée. Voici la signature de cette méthode :

```
// C#

public Object Add (string key, Object value, CacheDependency dependencies,
DateTime absoluteExpiration, TimeSpan slidingExpiration, CacheItemPriority
priority, CacheItemRemovedCallback onRemoveCallback)
```

```
' VB .NET

Public Function Add(ByVal key As String, ByVal value As Object, ByVal
dependencies As CacheDependency, ByVal absoluteExpiration As DateTime,
ByVal slidingExpiration As TimeSpan, ByVal priority As CacheItemPriority,
ByVal onRemoveCallback As CacheItemRemovedCallback) As Object
```

La signification des paramètres sont les suivants :

- *Key - Le nom de la variable de cache* : il permet d'identifier la variable dans le contexte de cache, et d'y accéder.
- *Value - La donnée mise en cache* : elle peut être de n'importe quel type.
- *Dependencies - Une dépendance de cache* : permet de créer une liaison entre les données mis en cache, et une ressource externe telle qu'un fichier ou une table d'une base de données. Spécifier *null* (C#) ou *nothing* (VB .NET) si aucune dépendance de cache ne doit être utilisée.
- *AbsoluteExpiration - Date d'expiration absolue* : date/heure à partir de laquelle la donnée sera automatiquement supprimée du cache. Si l'utilisation glissante est utilisée, alors spécifier *Cache.NoAbsoluteExpiration*.
- *SlidingExpiration - Date d'expiration glissante* : intervalle entre le dernier accès à l'objet ajouté au cache, et le moment où cet objet expire. Si l'expiration absolue est utilisée, alors spécifier *Cache.NoSlidingExpiration* (soit *TimeSpan.Zero*).
- *priority - La priorité* : priorité des éléments mis en cache
- *onRemoveCallback - Méthode de suppression de données dans le cache* : permet de spécifier une méthode, au travers d'un délégué, qui sera automatiquement exécutée, lorsque la variable de cache sera supprimée. Cette méthode doit respecter la signature du délégué *CacheItemRemovedCallback*, défini dans le Framework .NET :
  - Il s'agit d'une procédure.
  - Elle accepte trois paramètres d'entrée :
    - Le nom de la variable de cache (*string*).
    - Le contenu de la variable de cache (*object*).
    - Un objet de type *CacheItemRemovedReason*, fournissant une information sur le motif de la suppression : expiration, suppression explicite effectuée par l'application…

- En appliquant la méthode *Insert* sur l'objet *Cache*. Si une variable de cache de même nom existe déjà, alors cette variable est remplacée par la variable ajoutée. Elle propose différentes surcharges, la plus complète acceptant les mêmes paramètres que la méthode *Add*.

Voici un exemple d'utilisation, qui ajoute dans le cache, le contenu de la variable *sContenuFichier* pendant 15 secondes. La méthode *CacheItemRemoved* sera être automatiquement exécutée, lorsque cette variable de cache sera supprimée :

```
// C#

private void CacheItemRemoved(string key, object value,
CacheItemRemovedReason reason)
{
    string sMessage = string.Empty;

    sMessage += "clé : " + key;
    sMessage += "<br />";
    sMessage += "<br />";
    sMessage += "value : " + value.ToString();
    sMessage += "<br />";
    sMessage += "<br />";
    sMessage += "reason : " + reason.ToString();

    // Affichage du message.
    // ...
}


Cache.Insert("ContenuFichier", sContenuFichier, oDependanceCache,
DateTime.Now.AddSeconds(15), Cache.NoSlidingExpiration,
CacheItemPriority.Normal, new CacheItemRemovedCallback(CacheItemRemoved));
```

```
' VB .NET

Private Sub CacheItemRemoved(ByVal key As String, ByVal value As Object,
ByVal reason As CacheItemRemovedReason)
    Dim sMessage As String = String.Empty

    sMessage += "clé : " + key
    sMessage += "<br />"
    sMessage += "<br />"
    sMessage += "value : " + value.ToString()
    sMessage += "<br />"
    sMessage += "<br />"
    sMessage += "reason : " + reason.ToString()

    ' Affichage du message.
    ' ...
End Sub


Cache.Insert("ContenuFichier", sContenuFichier, oDependanceCache,
DateTime.Now.AddSeconds(15), Cache.NoSlidingExpiration,
CacheItemPriority.Normal, AddressOf CacheItemRemoved)
```

- Via un indexeur. Attention, cette méthode offre nettement moins de possibilité que les méthodes précédentes. Là aussi, si la variable de cache existe, alors son contenu est modifié. Elle est créée le cas échéant. Voici un exemple :

```
// C#
Cache["ListeClients"] = oListeClients;
```

```
' VB .NET
Cache("ListeClients") = oListeClients
```

2.2.3 Modification d'une donnée mise en cache

Pour modifier une donnée dans le cache, il suffit d'utiliser l'indexeur pour accéder et valoriser la variable de cache.

```
// C#
Cache["ListeClients"] = oNewListeClients;
```

```
' VB .NET
Cache("ListeClients") = oNewListeClients
```

2.2.4 Suppression d'une donnée du contexte de cache

Pour supprimer la donnée du cache, utiliser la méthode *Remove* sur l'objet *Cache* :

```
// C#
Cache.Remove("ListeClients");
```

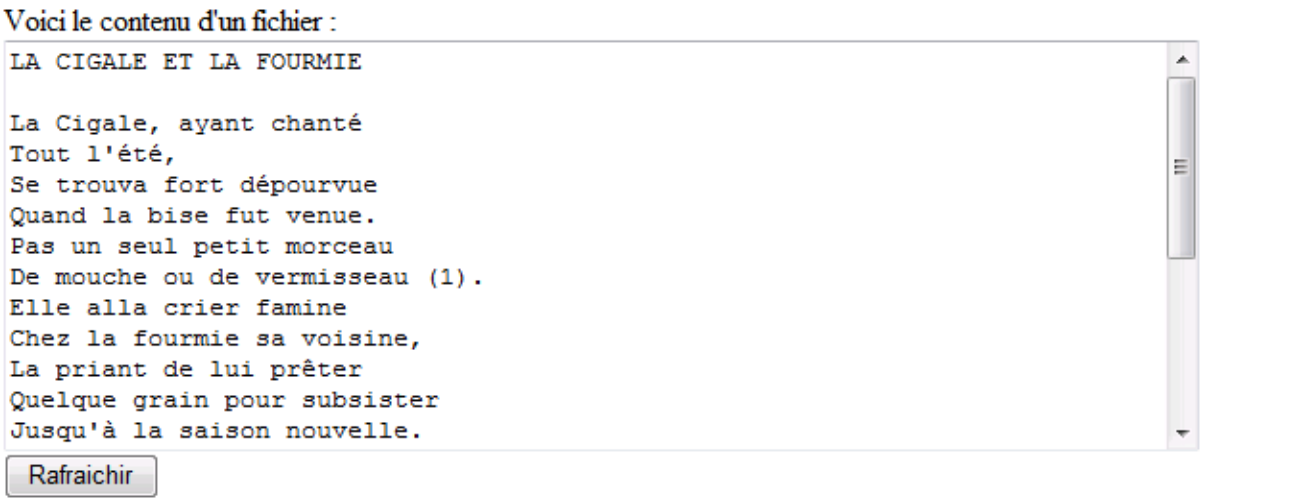
```
' VB .NET
Cache.Remove("ListeClients")
```

Remarque : il est interdit de valoriser à *null* (en C#) ou *Nothing* (en VB .NET) la variable de cache pour la détruire. Si vous écrivez cette instruction, alors une exception de type *System.ArgumentNullException* sera levée.

2.3 Un exemple concret de la mise de données en cache

Nous vous proposons d'étudier son utilisation au travers d'un exemple concret : le chargement et l'affichage du contenu d'un fichier.

Voici l'interface graphique de l'application :



Cette application :

- Permet de charger le contenu du fichier Data.Txt, contenu dans le répertoire *App_Data* de l'application ASP .NET.
- Affiche un message, de manière à confirmer que le fichier a été lu.

Voici le code source de l'application :

```
// C#

<form id="form1" runat="server">
    Voici le contenu d'un fichier :
    <br />
    <asp:TextBox ID="TxtContenuFichier" runat="server" ReadOnly="true"
TextMode="MultiLine" Columns="75" Rows="13"></asp:TextBox>
    <br />
    <asp:Button ID="CmdAfficherContenuFichier" runat="server"
Text="Rafraichir"
        onclick="CmdAfficherContenuFichier_Click" />
    <br />
    <br />
    Message : <asp:Label ID="LblMessageUtilisateur"
runat="server"></asp:Label>
</form>


protected void CmdAfficherContenuFichier_Click(object sender, EventArgs e)
{
    try
    {
        // Variables locales.
        string sNomCompletFichier;
        string sContenuFichier;

        // Initialisation.
        sNomCompletFichier = Server.MapPath("~/App_Data/Data.txt");
        sContenuFichier = string.Empty;

        // Chargement du contenu du fichier.
        sContenuFichier = System.IO.File.ReadAllText(sNomCompletFichier);

        // Affichage du contenu du fichier.
        TxtContenuFichier.Text = sContenuFichier;

        LblMessageUtilisateur.Text = "Le fichier a été lu à "+
DateTime.Now.ToString();
    }
    catch(Exception aEx)
    {
        LblMessageUtilisateur.Text = aEx.Message;
    }
}
```

```
' VB .NET

<form id="form1" runat="server">
    Voici le contenu d'un fichier :
    <br />
    <asp:TextBox ID="TxtContenuFichier" runat="server" TextMode="MultiLine"
Columns="75" Rows="13"></asp:TextBox>
    <br />
    <asp:Button ID="CmdAfficherContenuFichier" runat="server"
Text="Rafraichir" />
    <br />
    <br />
    Message : <asp:Label ID="LblMessageUtilisateur"
runat="server"></asp:Label>
</form>


Protected Sub CmdAfficherContenuFichier_Click(ByVal sender As Object, ByVal
e As System.EventArgs) Handles CmdAfficherContenuFichier.Click
    Try
        ' Variables locales.
        Dim sNomCompletFichier As String
        Dim sContenuFichier As String

        ' Initialisation.
        sNomCompletFichier = Server.MapPath("~/App_Data/Data.txt")
        sContenuFichier = String.Empty

        ' Chargement du contenu du fichier.
        sContenuFichier = System.IO.File.ReadAllText(sNomCompletFichier)

        ' Affichage du contenu du fichier.
        TxtContenuFichier.Text = sContenuFichier

        LblMessageUtilisateur.Text = "Le fichier a été lu à " +
DateTime.Now.ToString()
    Catch aEx As Exception
        LblMessageUtilisateur.Text = aEx.Message
    End Try
End Sub
```

Pour simuler le chargement d'un fichier de taille important, on stoppe le thread courant (chargé de l'exécution de la page ASP .NET) pendant 3 secondes. Ainsi, avant l'instruction du chargement du contenu du fichier, on ajoute la ligne indiquée en gras ci-dessous :

```
// C#

// Chargement du contenu du fichier.
System.Threading.Thread.Sleep(3000);
sContenuFichier = System.IO.File.ReadAllText(sNomCompletFichier);
```

```
' VB .NET

' Chargement du contenu du fichier.
System.Threading.Thread.Sleep(3000)
sContenuFichier = System.IO.File.ReadAllText(sNomCompletFichier)
```

Maintenant, à chaque chargement du fichier, l'utilisateur doit patienter 3 secondes. Pour améliorer les performances, nous allons utiliser le cache de données. Le principe de base est le suivant : on regarde si le contenu du fichier est déjà dans le cache. Pour ce faire, il suffit vérifier que la variable de cache, dans laquelle on aura le contenu du fichier existe. Deux cas sont alors possible :

- Il existe. Alors on le récupère et on l'affiche.
- Il n'existe pas. Alors on l'ajoute dans le cache et on l'affiche. On choisit une durée de vie absolue de 15 secondes.

Voici l'implémentation de cette fonctionnalité :

```
// C#

protected void CmdAfficherContenuFichier_Click(object sender, EventArgs e)
{
    try
    {
        // Variables locales.
        string sNomCompletFichier;
        string sContenuFichier;

        if (Cache["ContenuFichier"] == null)
        {
            // Initialisation.
            sNomCompletFichier = Server.MapPath("~/App_Data/Data.txt");
            sContenuFichier = string.Empty;

            // Chargement du contenu du fichier.
            System.Threading.Thread.Sleep(3000);
            sContenuFichier =
System.IO.File.ReadAllText(sNomCompletFichier);

            // Ajout des données dans le cache.
            Cache.Insert("ContenuFichier", sContenuFichier, null,
DateTime.Now.AddSeconds(15), TimeSpan.Zero);

            // Affichage du contenu du fichier.
            TxtContenuFichier.Text = sContenuFichier;

            LblMessageUtilisateur.Text = "Le fichier a été lu à " +
DateTime.Now.ToString();
        }
        else
        {
            // Affichage du contenu du fichier.
            TxtContenuFichier.Text = Cache["ContenuFichier"].ToString();

            LblMessageUtilisateur.Text = "La cache a été lu à " +
DateTime.Now.ToString();
        }
    }
    catch(Exception aEx)
    {
        LblMessageUtilisateur.Text = aEx.Message;
    }
}
```

```
' VB .NET

Protected Sub CmdAfficherContenuFichier_Click(ByVal sender As Object, ByVal
e As System.EventArgs) Handles CmdAfficherContenuFichier.Click
    Try
        ' Variables locales.
        Dim sNomCompletFichier As String
        Dim sContenuFichier As String

        If (Cache("ContenuFichier") Is Nothing) Then
            ' Initialisation.
            sNomCompletFichier = Server.MapPath("~/App_Data/Data.txt")
            sContenuFichier = String.Empty

            ' Chargement du contenu du fichier.
            System.Threading.Thread.Sleep(3000)
            sContenuFichier =
System.IO.File.ReadAllText(sNomCompletFichier)

            ' Ajout des données dans le cache.
            Cache.Insert("ContenuFichier", sContenuFichier, nothing,
DateTime.Now.AddSeconds(15), TimeSpan.Zero)

            ' Affichage du contenu du fichier.
            TxtContenuFichier.Text = sContenuFichier

            LblMessageUtilisateur.Text = "Le fichier a été lu à " +
DateTime.Now.ToString()
        Else
            ' Affichage du contenu du fichier.
            TxtContenuFichier.Text = Cache("ContenuFichier").ToString()

            LblMessageUtilisateur.Text = "La cache a été lu à " +
DateTime.Now.ToString()
        End If
    Catch aEx As Exception
        LblMessageUtilisateur.Text = aEx.Message
    End Try
End Sub
```

Maintenant, seulement lors du premier accès au fichier, l'utilisateur patiente trois secondes. Pendant les 15 secondes suivantes, l'utilisateur peut cliquer autant de fois qu'il souhaite sur le bouton « Rafraichir », il n'y aura pas de temps d'attente. Une fois ces quinze secondes passées, si l'utilisateur clique une nouvelle fois sur le bouton, alors il devra patienter de nouveau trois secondes…

Après avoir résolu le problème de performance lié au chargement du fichier, il subsiste toutefois un problème : si le fichier est modifié, renommé ou supprimé, le contenu affiché à l'écran reste le même, tant que le contenu du fichier est présent dans le cache de données. Les données affichées peuvent alors être considérées comme étant « impropres ». Une solution pour résoudre ce problème : utiliser une dépendance de cache.

3 Les dépendances de cache

3.1 Présentation

Les dépendances de cache représentent un mécanisme, permettant de gérer la fraîcheur des données, mises dans le cache de données d'une application ASP .NET. Le mécanisme consiste à lier une donnée mise en cache à :

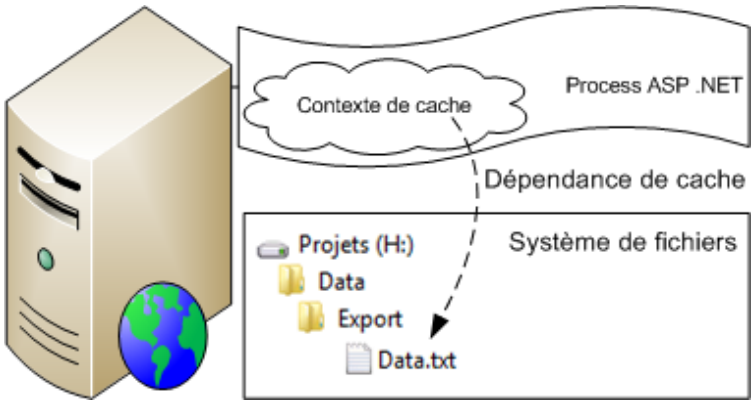
- Un fichier de données, texte ou binaire, situé sur le serveur ou dans un répertoire distant. Cette liaison permet que la donnée mise en cache, soit automatiquement supprimée dès que le fichier est modifié ou supprimé, quelque soit la durée de vie de la variable de cache.
- Une table d'une base de données. Le Framework .NET prend en charge nativement, uniquement les dépendances de cache vers les bases de données SQL Server 2000, 2005 et 2008). Cette liaison permet que la donnée mise en cache, soit automatiquement supprimée dès que la table est modifiée, quelque soit la durée de vie de la variable de cache.

Dans ce chapitre, nous étudierons la mise en œuvre d'une dépendance de cache, vers les deux sources de données citées ci-dessus. Cependant, il est possible de créer une dépendance de cache personnalisée, en créant une classe dérivant de la classe *System.Web.Caching.CacheDependency*.

3.2 Les dépendances de cache vers un fichier

3.2.1 Présentation

Une dépendance de cache vers un fichier, permet de lier une donnée à un fichier présent dans le système de fichier de serveur, ou distant (à condition d'avoir tous les droits requis).



Dans le schéma ci-dessus, toute opération de suppression, modification du nom ou du contenu du fichier, entraîne de manière automatique la suppression de la variable de cache liée à ce fichier.

3.2.2 Mise en œuvre

Pour créer une dépendance de cache vers un fichier, il est nécessaire d'utiliser la classe *System.Web.Caching.CacheDependency* du Framework .NET. Dans le chapitre précédent, nous avions besoin de créer une dépendance de cache, entre le fichier *Data.*Txt et son contenu situé dans le contexte de cache de l'application. Les modifications à apporter au code de l'exercice sont les suivantes :

- Création d'une dépendance de cache, en utilisant le nom complet du fichier *Data.Txt*.
- Utilisation de cette dépendance de cache, lors de l'ajout du contenu de fichier dans le contexte de cache.

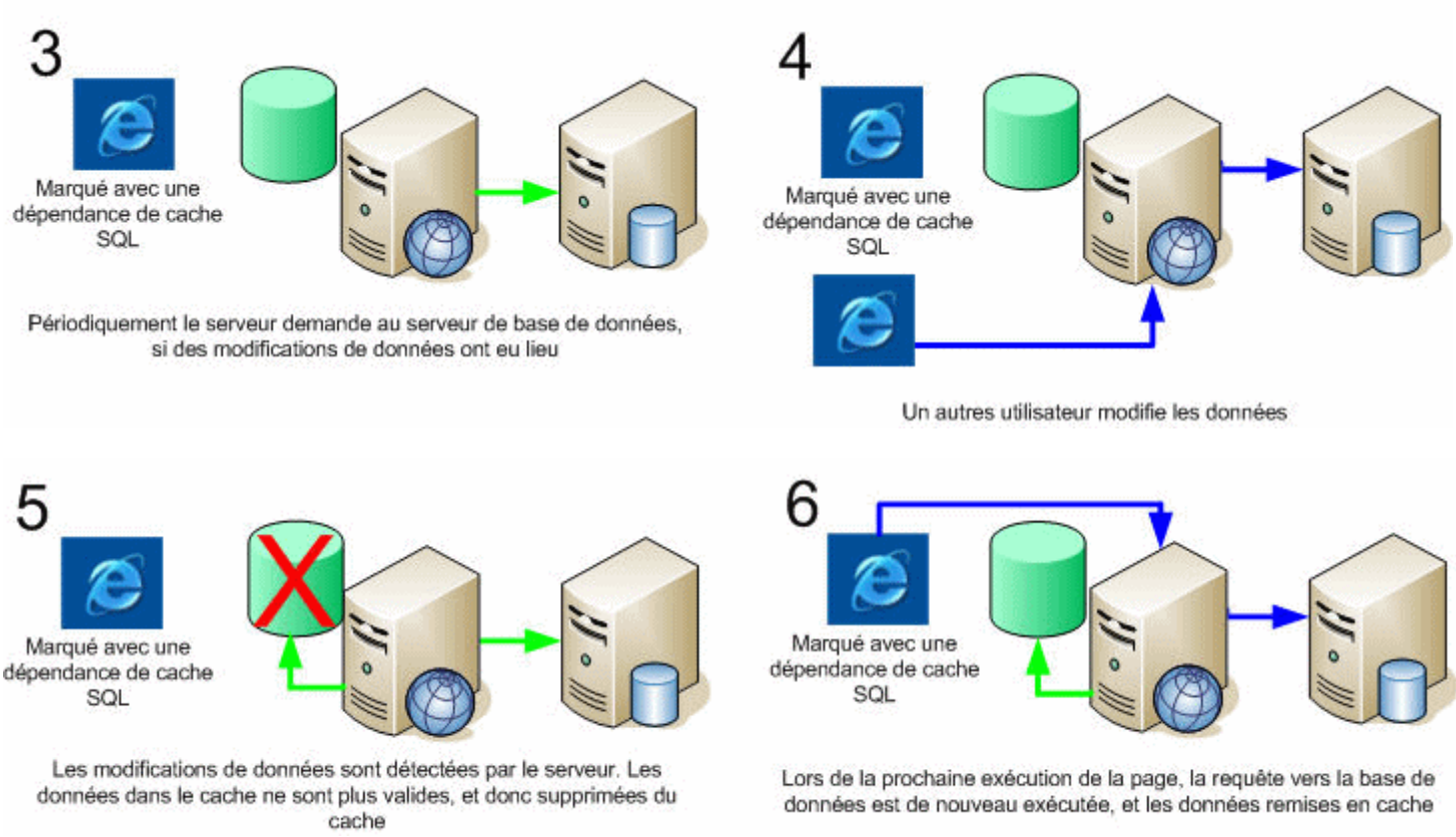
```
// C#

// Création de la dépendance de cache vers le fichier.
CacheDependency oDependanceCache = new CacheDependency(sNomCompletFichier);

// Ajout des données dans le cache.
Cache.Insert("ContenuFichier", sContenuFichier, oDependanceCache,
DateTime.Now.AddSeconds(15), TimeSpan.Zero);
```

```
' VB .NET

' Création de la dépendance de cache vers le fichier.
Dim oDependanceCache As New CacheDependency(sNomCompletFichier)

' Ajout des données dans le cache.
Cache.Insert("ContenuFichier", sContenuFichier, oDependanceCache,
DateTime.Now.AddSeconds(15), TimeSpan.Zero)
```

3.3 Les dépendances de cache vers une table vers une base de données SQL Server

3.3.1 Présentation

Voici un schéma décrivant le fonctionnement d'une dépendance de cache, entre une donnée mise en cache, et une table d'une base de donnes SQL Server 2000 :



Avec SQL Server 2005, l'envoi d'une notification à destination du serveur IIS, est prise en charge par lui-même. En conséquence, l'étape 3 n'existe pas, et dans l'étape 5, le sens de la flèche du serveur IIS vers le serveur de base de données est inversé.

3.3.2 Mise en œuvre

Les étapes de mise en œuvre d'une dépendance de cache sont les suivantes :

- Préparation de la base de données ;
- Configuration de l'application ;
- Création et utilisation de la dépendance de cache.

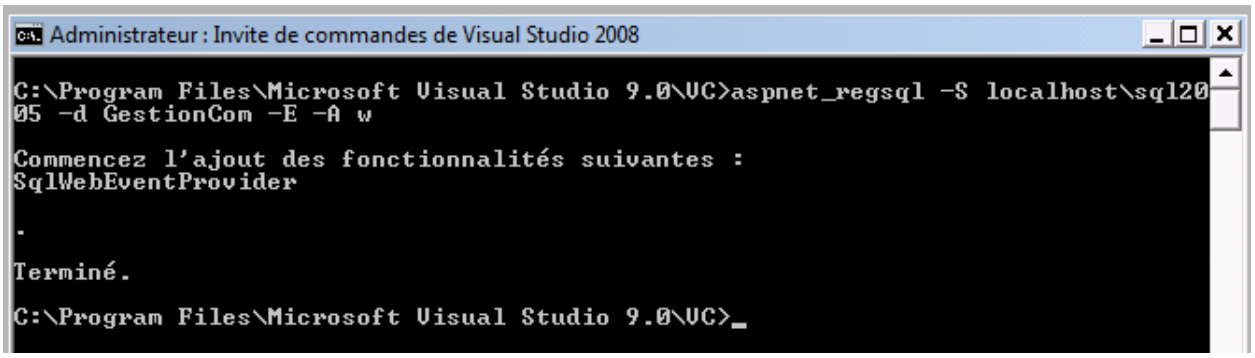
Nous allons créer une dépendance de cache sur une table nommée *Societe*, contenu dans une base de données SQL Server 2005, nommée *GestionCom*.

3.3.2.1 Préparation de la base de données

Pour préparer la base de données *GestionCom*, afin de créer dans nos applications ASP .NET, des dépendances de cache vers la table *Societe*, il suffit d'utiliser l'outil *aspnet_regsql* (fourni avec le Framework .NET), dans une fenêtre de commandes DOS pour Visual Studio. Nous vous invitons à consulter l'aide de cet outil sur MSDN, qui est un véritable couteau-suisse, par les multiples fonctionnalités et options qu'il propose. Dans notre cas, nous utiliserons la commande suivante :

```
aspnet_regsql -S localhost\sql2005 -d GestionCom -E -A w
```

Voici un exemple d'utilisation de cette commande :

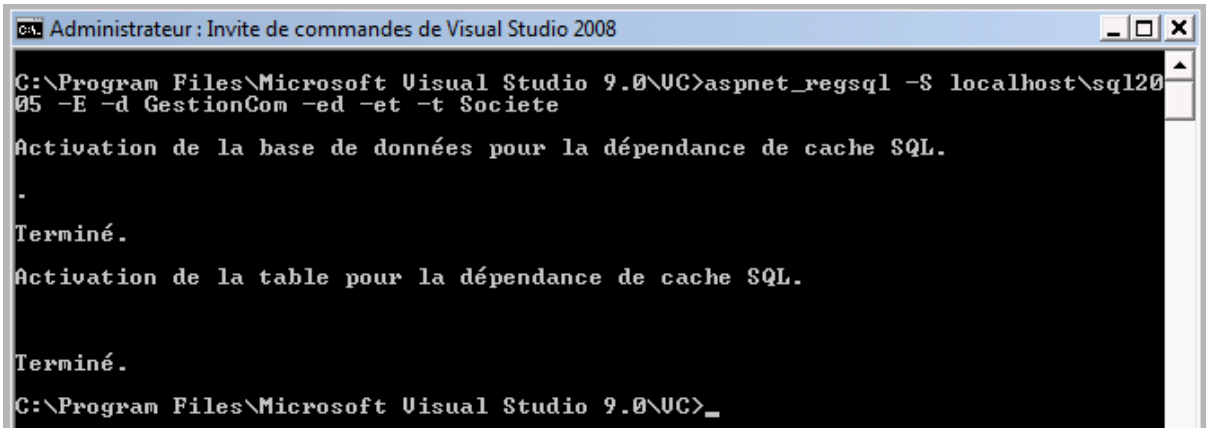


L'exécution de cette commande a pour conséquences de créer de nombreux objets SQL (tables, vues et procédures stockées) dans la base de données *GestionCom*, afin de générer les notifications de cache SQL.

Puis, pour permettre à SQL Server de lever des notifications de cache sur la table *Societe* de cette base de données, lorsque des données y sont ajoutées, modifiées ou supprimées, il est nécessaire d'exécuter dans la même console DOS, l'instruction suivante :

```
aspnet_regsql -S localhost\sql2005 -E -d GestionCom -ed -et -t Societe
```

Voici un exemple d'utilisation de cette commande :



L'exécution de cette instruction a pour conséquences :

- On remarque la création de la table *AspNet_SqlCacheTablesForChangeNotification*, qui enregistre les tables de la base de données, sur lesquelles les notifications de cache pourront être activées. Ainsi la table *Societe* est enregistrée dans cette même table. Et pour chacune des tables, un indicateur incrémental est géré, utilisé par SQL Server pour détecter les tables modifiées, et lever ainsi les notifications de cache SQL.

- Un trigger (déclencheur) est créé sur la table *Societe*. Il se nomme *Societe_AspNet_SqlCacheNotification_Trigger*, et son implémentation est la suivante :

```
CREATE TRIGGER [Societe_AspNet_SqlCacheNotification_Trigger] ON
[dbo].[Societe]
     FOR INSERT, UPDATE, DELETE AS BEGIN
     SET NOCOUNT ON
     EXEC dbo.AspNet_SqlCacheUpdateChangeIdStoredProcedure N'Societe'
END
```

On peut observer que ce trigger exécute la procédure stockée *SqlCacheUpdateChangeIdStoredProcedure*, en passant en paramètre le nom de la table. Cette procédure stockée permet de gérer l'indicateur incrémental. Voici le code de cette procédure stockée :

```
CREATE PROCEDURE [dbo].[AspNet_SqlCacheUpdateChangeIdStoredProcedure]
     @tableName NVARCHAR(450) AS
BEGIN
     UPDATE dbo.AspNet_SqlCacheTablesForChangeNotification WITH (ROWLOCK)
SET changeId = changeId + 1 WHERE tableName = @tableName
END
```

Ainsi, toutes les applications abonnées via une dépendance de cache vers la table *Societe*, seront automatiquement averties, et de manière transparente.

3.3.2.2 Configuration de l'application

Dans le fichier de configuration de l'application ASP .NET :

- Ajoutons dans l'élément *connectionStrings*, une chaîne de connexion, contenant toutes les informations nécessaires permettant de se connecter à la base de données *GestionCom* :

```
<connectionStrings>
  <add name="GestionCom_CS" connectionString="Data
                Source=localhost\sql2005; Initial Catalog=GestionCom;
                user id=userGestionCom; password=passwd;"/>
</connectionStrings>
```

- Puis, déclarons une dépendance de cache SQL vers la base de données. Sous l'élément *system.web*, ajoutons les éléments suivants :

```
<caching>
  <sqlCacheDependency enabled="true">
    <databases>
      <add name="SqlCaching" connectionStringName="GestionCom_CS" />
    </databases>
  </sqlCacheDependency>
</caching>
```

3.3.2.3 Création et utilisation de la dépendance de cache

Dans une page de l'application, créons une dépendance de cache vers la table *Societe*. Il existe deux manières de la réaliser :

- De manière déclarative, dans le XHTML de la page :

```
<asp:SqlDataSource ID="SqlDsSocietes" runat="server"
    ConnectionString="<%$ ConnectionStrings:GestionCom_CS %>"
    SelectCommand="SELECT Identifiant, RaisonSociale FROM Societe"
    EnableCaching="True" SqlCacheDependency="GestionCom:Societe">
</asp:SqlDataSource>

<asp:GridView ID="LstSocietes" runat="server" AutoGenerateColumns="False"
    DataKeyNames="Identifiant" DataSourceID="SqlDsSocietes">
    <Columns>
        <asp:BoundField DataField="Identifiant" HeaderText="Id"
ReadOnly="True" />
        <asp:BoundField DataField="RaisonSociale" HeaderText="Raison
Sociale" />
    </Columns>
</asp:GridView>
```

- De manière impérative, en C# ou VB .NET :

```
// C# et VB .NET

<asp:GridView ID="LstSocietes1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="Identifiant">
    <Columns>
        <asp:BoundField DataField="Identifiant" HeaderText="Id"
ReadOnly="True" />
        <asp:BoundField DataField="RaisonSociale" HeaderText="Raison
Sociale" />
    </Columns>
</asp:GridView>
```

```
// C#

// Variables locales.
SqlDataAdapter oSqlDataAdapter;
DataTable oTableSocietes;

if (Cache["TableSociete"] == null)
{
    // Pour simuler le chargement de données volumineuses.
    System.Threading.Thread.Sleep(3000);

    // Création de la source de données.
    oSqlDataAdapter = new SqlDataAdapter("SELECT Identifiant, RaisonSociale
FROM Societe",
ConfigurationManager.ConnectionStrings["GestionCom_CS"].ConnectionString);
    oTableSocietes = new DataTable("SOCIETE");
    oSqlDataAdapter.Fill(oTableSocietes);

    // Création de la dépendance dans le cache.
    SqlCacheDependency oCacheDependance;
    oCacheDependance = new SqlCacheDependency("GestionCom", "Societe");

    // Ajout de la table dans le cache.
    Cache.Insert("TableSociete", oTableSocietes, oCacheDependance,
DateTime.Now.AddSeconds(60), TimeSpan.Zero);
}
else
{
    // Lecture des données du cache.
    oTableSocietes = (DataTable)Cache["TableSociete"];
}

// Définition de la source de données de la grille (DataBinding).
LstSocietes1.DataSource = oTableSocietes;

// Affichage des données.
LstSocietes1.DataBind();
```

```vb
' VB .NET

' Variables locales.
Dim oSqlDsSocietes As SqlDataAdapter
Dim oTableSocietes As DataTable

If (Cache("TableSociete") Is Nothing) Then
    ' Pour simuler le chargement de données volumineuses.
    System.Threading.Thread.Sleep(3000)

    ' Création de la source de données.
    oSqlDsSocietes = New SqlDataAdapter("SELECT Identifiant, RaisonSociale
FROM Societe",
ConfigurationManager.ConnectionStrings("GestionCom_CS").ConnectionString)
    oTableSocietes = New DataTable("SOCIETE")
    oSqlDsSocietes.Fill(oTableSocietes)

    ' Création de la dépendance dans le cache.
    Dim oCacheDependance As SqlCacheDependency
    oCacheDependance = New SqlCacheDependency("GestionCom", "Societe")

' Ajout de la table dans le cache.
    Cache.Insert("TableSociete", oSqlDsSocietes, oCacheDependance,
DateTime.Now.AddSeconds(60), TimeSpan.Zero)
Else
    ' Lecture des données du cache.
    oTableSocietes = CType(Cache("TableSociete"), DataTable)

    ' Définition de la source de données de la grille (DataBinding).
    LstSocietes1.DataSource = oTableSocietes

    ' Affichage des données.
    LstSocietes1.DataBind()
End If
```

4 Mise en cache des pages et des parties de page

4.1 Présentation

Pour les pages Web dites « déterministes » de votre application ASP .NET, vous pouvez demander à les stocker dans le cache ASP .NET, afin d'améliorer les performances de votre application. Une page « déterministe » est une page dont le contenu ne varie pas, voire peu, lorsqu'elle est exécutée plusieurs fois dans le temps. Par exemple, vous avez conçu une page, qui affiche les statistiques de vente des commerciaux du mois précédent. Le calcul de ces statistiques constitue un temps de traitement non négligeable, et sont eux-mêmes dits « déterministes ».

Deux manières en mettre en cache les résultats d'exécution d'une même page ASP .NET :

- De manière déclarative, en utilisant la directive *OutputCache* dans les pages ASP .NET.
- De manière impérative, en utilisant la classe *HttpCachePolicy*.

4.2 Mise en cache de page

4.2.1 Utilisation de la directive *OutputCache*

Dans une page ASP .NET, La directive *OutputCache* permet de mettre en cache plusieurs résultats d'exécution de cette même page. Les attributs de cette directive sont les suivants :

Attribut	Signification	Obligatoire
VaryByParam	Permet d'enregistrer le résultat d'exécution d'une page en fonction de la valeur d'un paramètre dans l'URL. Si aucun paramètre ne doit être spécifié, spécifier alors la valeur *None*	oui
VaryByControl	Permet d'enregistrer le résultat d'exécution d'une page en fonction de la valeur d'un contrôle de la page	oui
VaryByHeader	Permet d'enregistrer le résultat d'exécution d'une page en fonction de l'entête de la requête HTTP	
VaryByCustom	Permet d'enregistrer le résultat d'exécution d'une page en fonction du type du navigateur ou d'une donnée personnalisée	
Duration	Permet de définir en secondes, la durée de vie absolue de la mise en cache	
OutputCacheLocation	Peut posséder les valeurs suivantes : *Any* (par défaut), *Client*, *Downstream*, *Server*, *ServerAndClient* ou *None*.	

Voici un exemple : créons une page, permettant à l'utilisateur d'envoyer une requête HTTP, en passant définissant un paramètre de requête (*id*), valorisé avec une valeur fournie par l'utilisateur. Lors de l'exécution de la page, on inscrit la date et l'heure courante.

04/01/2009 11:50:53

Identifiant : 1

Rafraichir

Voici le code source de cette page :

```
// C# et VB .NET

<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="OutputCache.aspx.cs" Inherits="OutputCache" %>

<!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></title>

    <script type="text/javascript">
        function Valider() {
            form1.action = 'OutputCache.aspx?id=' +
document.getElementById('TxtIdentifiantRequete').value;
            form1.submit();
        }
    </script>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="LblDateHeureCourante" runat="server"
Text="Label"></asp:Label>
            <br />
            <br />
            <br />
            Identifiant : <asp:TextBox ID="TxtIdentifiantRequete"
runat="server"></asp:TextBox>
            <br />
            <br />
            <input id="CmdValider" type="button" value="Valider"
onclick="Valider();" />
        </div>
    </form>
</body>
</html>
```

```
// C#

public partial class OutputCache : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(3000);
        LblDateHeureCourante.Text = DateTime.Now.ToString();
    }
}
```

```
// VB .NET

Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    System.Threading.Thread.Sleep(3000)
    LblDateHeureCourante.Text = DateTime.Now.ToString()
End Sub
```

On peut alors saisir la valeur de différents identifiants, et cliquer sur le bouton *Valider*. Pour mieux observer l'utilisation future de la mise en cache des pages, l'exécution de la page est volontairement ralenti de 3 secondes :

Identifiant saisie	*Date / heure affichée*
1	04/01/2009 11:50:53
2	04/01/2009 12:08:35
3	04/01/2009 12:08:47
4	04/01/2009 12:09:11
5	04/01/2009 12:09:24

Comme la mise en cache du résultat d'exécution de la page n'est pas activée, si on valide la page avec deux fois le même identifiant, alors l'heure affichée sera différente. Pour l'activer, ajoutons la directive *OutputCache* dans cette même page. Voici un exemple d'utilisation, permettant de mettre différentes versions du résultat d'exécution de la page en cache, en fonction de la valeur du paramètre de requête *Id* :

```
// C# et VB .NET

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

Ainsi, pour deux requêtes HTTP envoyées avec moins de 30 secondes d'intervalle, avec le même identifiant :

- La première exécution : 3 secondes d'attente, puis le résultat est mis en cache ;
- La seconde exécution : la page n'est pas exécutée côté serveur. Le résultat précédemment mis en cache est lu, et directement renvoyé au client.

4.2.2 Utilisation de la classe *HttpCachePolicy*

L'objet ASP .NET *Response* possède une propriété *Cache*, de type *HttpCachePolicy*. Cette classe permet de gérer le cache de sortie des pages, de manière impérative. Pour en savoir plus sur cette page, nous vous invitons à vous rendre sur le site MSDN, à l'adresse suivante :

http://msdn.microsoft.com/fr-fr/library/system.web.httpcachepolicy.aspx

4.3 Mise en page de parties de pages

Pour mettre en page une partie d'une page, il est possible d'utiliser les contrôles utilisateurs Web. Voici un exemple de ce type de contrôle, dont le but est d'afficher la date et l'heure courante :

```
// C#

<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="UC_Commandes.ascx.cs" Inherits="UC_Commandes" %>

Date de mise à jour des données : <asp:Label ID="LblDateHeureCourante"
runat="server"></asp:Label>



protected void Page_Load(object sender, EventArgs e)
{
    LblDateHeureCourante.Text = DateTime.Now.ToString();
}
```

```
' VB .NET

<%@ Control Language="VB" AutoEventWireup="false"
CodeFile="UC_Commandes.ascx.vb" Inherits="UC_Commandes" %>

Date de mise à jour des données : <asp:Label ID="LblDateHeureCourante"
runat="server"></asp:Label>



Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    LblDateHeureCourante.Text = DateTime.Now.ToString()
End Sub
```

Ajoutons ce contrôle ce contrôle dans une page ASP .NET :

```
// C# ET VB .NET

<%--Ajout d'une directive référençant le contrôle Utilisateur Web--%>
<%@ Register src="UC_Commandes.ascx" tagname="UC_Commandes" tagprefix="uc1"
%>

<%--Dans le corps de la page, ajout du contrôle Utilisateur Web--%>
<uc1:UC_Commandes ID="UC_Commandes1" runat="server" />
```

Lors de différentes exécutions successives de la page, la date et l'heure courante est affichée. Pour enregistrer une version du contrôle utilisateur Web en cache, il est nécessaire d'ajouter la directive *OutputCache* :

- Dans le contrôle utilisateur lui-même. Dans ce cas, la mise en cache est centralisée dans le contrôle Utilisateur Web, et sera valable pour toutes ses utilisations dans les pages de l'application :

```
// C# ET VB .NET

<%@ OutputCache duration="10" varybyparam="none" %>
```

- Dans la page ASP .NET. Dans ce cas, seule l'instance du contrôle Utilisateur Web (visée par l'attribut *VaryByControl*) sera mise en cache :

```
// C# ET VB .NET

<%@ OutputCache duration="10" varybyparam="none"
VaryByControl="UC_Commandes1" %>
```

Il est possible d'utiliser les deux mises en cache. Dans ce cas, c'est la durée de mise en cache la plus longue qui est prise en compte.

4.4 Utilisation du contrôle ASP .NET Substitution

Dans la version de la page ou d'une partie d'une page mise en cache, il peut exister une information qu'on souhaite mettre à jour, avant de renvoyer la page au client. ASP .NET propose alors de substituer le contenu mis en cache, au travers de l'utilisation du contrôle *Substitution*.

Par exemple, dans l'exemple précédent concernant la mise en cache de la page, nous souhaitons afficher la date et l'heure de lecture de la page dans le cache d'exécution des pages, avant de renvoyer la page au client (autrement dit modifier le contenu mis en cache). Dans notre page ASP .NET, ajoutons le contrôle *Substitution*. Puis, valorisons la propriété *MethodName*, avec une méthode contenue dans la classe code-behind de la page, qui respecte la signature du délégué *HttpResponseSubstitutionCallback* du Framework .NET :

```
// C# et VB .NET

Date et heure de lecture de la page dans le cache d'exécution des pages :
<asp:Substitution ID="LblDateHeureCache" runat="server"
MethodName="AfficherDateHeureLectureCache" />
```

```
// C#

public static string AfficherDateHeureLectureCache(HttpContext aContext)
{
    return DateTime.Now.ToString();
}
```

```
' VB .NET

Public Shared Function AfficherDateHeureLectureCache(ByVal aContext As
HttpContext) As String
    Return DateTime.Now.ToString()
End Function
```

Comme nous pouvons le constater, la méthode *AfficherDateHeureLectureCache* retourne la date et l'heure courante, qui sera affichée en lieu et place du contrôle de substitution. Ses caractéristiques sont les suivantes (liées à la signature du délégué *HttpResponseSubstitutionCallback*) :

- Elle est statique.
- Elle retourne une chaîne de caractères.
- Elle accepte un seul paramètre, de type *HttpContext*. Comme le nom du type l'indique, ce paramètre expose dans le corps de la méthode le contexte Http courant, via des propriétés. En voici quelques unes :
    - Application : accès au contexte d'application.
    - Cache : accès au contexte de cache.
    - Session : accès au contexte de session.
    - Request : accès au contexte de requête.
    - …

5 Conclusion

Ce cours vous a montré différentes facettes, de l'utilisation du cache dans les applications ASP .NET, et des plus values que cette utilisation pouvait apporter, notamment dans l'amélioration des performances d'exécution des pages.

Toutefois, il est important de retenir que toute utilisation d'un cache de données, nécessite d'effectuer des copies de données mises en cache. Il revient alors à l'utilisateur de gérer la fraîcheur des données, de manière à ce que les applications n'utilisent et n'affichent pas des données « impropres ». Les dépendances de cache proposées par la technologie ASP .NET, apportent une réponse pour gérer cette fraîcheur des données.