Gestion de l'état dans une application ASP.NET

par Nicolas Barlatier

Date de publication : 20 Août 2006

Dernière mise à jour : 20 Août 2006

Gestion de l'état dans une application ASP.NET

I - Introduction

Le but de ce tutorial est de vous donner une bonne compréhension de toutes les options mises à votre disposition pour gérer l'état d'une page web. Les valeurs des variables et des contrôles constituent l'état d'une page web. La gestion de l'état représente le traitement de maintenir l'état d'une page web à travers plusieurs aller-retours avec le serveur lors de la navigation. A cause de la nature déconnectée d'HTTP, la gestion de l'état représente un problème important dans les applications Web. ASP.NET fournit plusieurs techniques pour conserver les informations sur l'état (state)à travers plusieurs postbacks de la page. On peut classer ces techniques en deux catégories : des techniques côté client et des techniques côté serveur, cela dépends d'où sont consommées les ressources pour la gestion de l''état. J'espère que ce cours vous aidera à mieux savoir que choisir comme option pour conserver une certaine valeur.

II - Techniques du côté client pour la gestion de l'état d'une page web

Les techniques du côté client le code HTML et les fonctionnalités de l'exploreur web pour stocker les informations liées à l'état. ASP.NET met à votre disposition les techniques suivantes pour stocker les informations de l'état du côté client:

- Query strings
- Cookies
- Les champs cachés

Nous allons les examiner en détails afin de bien discerner les différences et avantages/inconvénients.

II-A - Les Query strings (Les chaînes passées dans la requête dans l'URL)

Les Query strings sont utilisées pour maintenir l'état en ajoutant à la fin de l'URL de la page l'information sur l'état (State). Les données de l'état sont séparées de l'URL même par un marqueur sous la forme d'un point d'interrogation (?). Les données rattachées à l'URL sont habituellement sous la forme d'un ensemble de paires clé-valeur où chaque paire clé-valeur est séparée par un ET commercial ou un ampersand (&). Par exemple, examinez l'URL qui contient des paires clé-valeur :

http://irnbru.org/enter.aspx?name=nicolas+barlatier&position=Engineer

Les query strings sont souvent utilisés pour faire passer des informations de petite taille vers des pages web. Les query strings ont cependant des limites qui sont les suivantes:

- Des exploreurs limitent la longueur du query string, cela limite donc la quantité de données que vous pouvez passer dans l'URL
- Les query strings ne prennent pas en charge les types de données structurés
- Les données passées par les query strings ne sont pas sécurisées car elles sont facilement visibles dans la zone de saisie de l'URL de l'exploreur

Lire les informations contenues dans les query strings en ASP.NET se fait bien facilement à l'aide de la propriété QueryString contenu dans l'objet HttpRequest en cours (objet représentant la requête en cours). La propriété QueryString retourne un objet NameValueCollection qui représente les paires clé-valeur stockées dans le query string. Par exemple, les instructions suivantes récupèrent le nom de la personne et son poste dans la compagnie à partir du query string utilisé dans l'URL d'exemple ci-dessus:

Request.QueryString["name"]

Request.QueryString["position"]

II-B - Les cookies

Les Cookies représentent des paquets d'information. Chaque paquet d'information est stocké comme des paires clé-valeur du côté client. Ces paquets sont associés avec un domaine spécifique et sont envoyés avec chaque requête vers le serveur web associé. Les cookies sont souvent utilisés pour garder les préférences d'un utilisateur et pour leur fournir un service personnalisé lors de leurs visites suivantes vers une page Web. L'utilisation des

cookies présente l'ensemble des limites suivant:

- La plupart des exploreurs limitent la taille des informations que vous pouvez stocker à l'intérieur d'un cookie. La taille est habituellement 4,096 octets pour les plus anciens exploreurs et 8,192 octers pour les versions plus récentes.
- Les utilisateurs peuvent refuser les cookies
- Même si vous demandez à l'explorateur de garder un cookie sur le pc d'un utilisateur pendant une période donnée, l'exploreur peut passer outre de cette demande en imposant sa propre limite d'expiration
- Parce que les cookies sont stockés chez le client, ils peuvent être modifiés. Vous ne devez pas faire confiance aux données que vous recevez d'un cookie

La propriété Cookies de l'objet HttpRequest vous renvoit un objet HttpCookieCollection qui représente l'ensemble de cookies envoyés par le client lors de la requête en cours.

Le code suivant vous explique comment initialiser un cookie sur l'ordinateur de l'utilisateur:

Créer un cookie sur le PC de l'utilisateur
```
// Création du cookie "Utilisateur"
HttpCookie cUtilisateur = new HttpCookie("Utilisateur");
cUtilisateur.Value = "Nicolas;

// Mettre le temps d'expiration à 15 min à partir de maintenant
cUtilisteur.Expires = DateTime.Now + new TimeSpan(0,0,15,0);

// Ajouter le cookies à la collection de cookies de la réponse (à la requête de l'utilisateur)
// Il sera envoyé sur la machine de l'utilisateur
Response.Cookies.Add(cName);
```

Pour récupérer cette valeur dans le cookie, vous pouvez la récupérer simplement dans la requête HTTP comme dans l'exemple de code suivant :

Récupérer une valeur contenue dans un cookie
```
if(Request.Cookies["Utilisateur"] == null)
{

}
else
{

   Response.Write(Request.Cookies["Utilisateur"].Value);
}
```

II-C - Les champs cachés

Les champs cachés contiennt des informations qui ne sont pas visible sur la page mais qui sont postées vers le serveur lors d'un postback de la page. Tous les exploreurs modernes prennent en charge lec champs cachés d'une page Web. Mais les champs cachés présentent les limites suivantes :

- Bien qu'une information stockée dans un champ caché ne soit pas visible sur la page, il fait encore parti du code HTML de la page et les utilisateurs peuvent visualiser la valeur d'un champs caché avec code source HTML d la page. Les champs cachés ne représentent donc pas un bon chois pour stocker des informations que vous voulez garder sécuriser
- Les champs cachés font parti de la page HTML. Si vous stocker plus d'informations dans des champs cachés, cela augmentera la taille de la page HTML, ce qui rendra le téléchargement de la page plus long pour l'utilisateur
- Les champs cachés vous permettent de ne stocker qu'une seule valeur dans un champs. Si vous souhaitez stocker des valeurs structurées vous devez utiliser plusierus champs cachés

ASP.NET met à disposition un contrôle serveur HTML: HtmlInputHidden qui correspond à l'élement HTML <input type="hidden">. Le contrôle HtmlInputHidden n'est pas disponible en tant que contrôle serveur Web parce que ASP.NET utilise une technique similaire mais plus puissante: le view state.

II-D - Le View State

View state est le mécanisme qu'ASP.NET utilise pour maintenir l'état des contrôles à traveurs plusieurs postabacks d'une page. Tout comme les champs cachés et les cookies, vous pouvez aussi utiliser le view state pour maintenir l'état des valeurs non liées aux contrôles d'une page. Cependant il est important de remarquer que le view state ne marche que lorsque la page a subit un post back vers elle même.

Vous avez plusieurs possibilités.

II-D-1 - Le view state pour les contrôles postback

Quelques contrôles comme TextBox, CheckBox, et ainsi de suite postent leur valeurs lors de l'opération postback. Ces types de contrôles sont connus sous le nom de contrôles postback. Pour les contrôles postback, ASP.NET récupère leurs valeurs une par une à partir de la requête HTTP et les copie vers la valeur du contrôle lors de la création de la réponse HTTP. Dans le passé, les développeurs Web devaient écrire à la main ce code pour maintenir l'était des contrôles postback, mais à présent ASP.NET le fait pour vous automatiquement.

> *Ces contrôles postback ne récupèrent donc pas leurs valeurs à partir du view state. Il s'agit d'une erreur commune d'assumer le contraire! Ces valeurs sont récupérées directement à partir de la requête et non du view state.*

II-D-2 - Le View State pour les contrôles Nonpostback

Le mécanisme view state d'ASP.NET conserve les valeurs des contrôles nonpostback controls (c'est à dire les contrôles qui ne postent pas leurs valeurs lors de l'opération postback comme un contrôle Label). ASP.NET parvient à maintenir les valeurs d'un contrôle même lorsque que les contrôles ne postent pas ses valeurs. En fait ASP.NET étend le concept de champs cachés.

Quand ASP.NET execute une page, il récupère les valeurs de tous les contrôles nonpostback qui sont modifiées dans le code et les formate en un seul string encodé avec le format base64. Ce string (chaîne de caractères) est alors stocké dans un champs caché and un contrôle appelé __VIEWSTATE :

```
<input type="hidden" name="__VIEWSTATE" value=
"dDwtMTg3NjA4ODA2MDs7PoYLsizcOhkv2XeRfSJNPt12o1HP" />
```

Le champs d'entrée (input) caché est un contrôle postback donc lors du prochain postback de la page, le string encodé dans le champ __VIEWSTATE est aussi envoyé dans le post. Sur le serveur Web, ASP.NET décode le string view state à l'initialisation de la page et restaure les valeurs des contrôles contenus dans la page.

Maintenir un état en utilisant cette technique ne demande pas beaucoup de ressources sur le serveur mais cela

augment la taille du fichier HTML, ce qui peut augmenter le temps pour charger la page.

II-D-3 - Le View State pour les valeurs globables de la page

La propriété ViewState de la classe Page représente un bon endroit pour stockée des valeurs globales de la page. View state sauve ces valeurs juste avant de rendre la page et restaure les valeurs au moment de l'initialisation de la page après une opération postback. Cela peut ressembler à des cookies ou des champs cachés, mais la grande amélioration est que vous n'êtes pas limité à stocker des valeurs simples. Vous pouvez utiliser le ViewState pour stocker tout objet tant qu'il soit sérialisable.

Donc le view state n'est pas limité aux contrôles serveur, votre code de la page web peut ajouter des informations directement dans la collection de la page conteneur puis les récupérer par la suite après que la page est été renvoyée avec postback. Le types d'information que vous pouvez stocker incluent les types de données simples et vos propres objets personnalisés.

La propriété **ViewState** de la page vous fournis un accès à la collection du view state. Cette propriété renvoit une instance de la classe collection **StateBag**. Pour ajouter et enlever des éléments de cette classe, vous utilisez la syntaxe base sur le dictionnaire (dictionnary) où chaque élément possède un nom string unique

Utilisation de la propriété ViewState de la classe Page

```
// Le mot clé this représente l'objet du type Page en cours, il n'est pas obligatoire
this.ViewState["Compteur"] = 1;
```

Le code ci-dessus place un entier contenant la valeur 1 dans la collection du state view et donne le nom descriptif Compteur. Si aucun élément ne possède ce nom, un nouvel élément sera automatiquement ajouté. Si un élément est déjà indexé sous ce nom, il sera mis à jour.

Lors de la récupération d'une valeur, vous utiliser le nom de la clé. Vous devez aussi transtyper la valeur récupérée vers le type de donnée approprié en utilisant la syntaxe de transtypage (cast).

Cette étape est nécessaire car la collection du ViewState stocke tous les éléments comme des objects génériques. Ce qui permet de gérer beaucoup de types de données différents.

L'exemple suivant vous explique comment procéder :

Comment ajouter et utiliser une variable globale

```
// Nombre de fois une page est visualisée
protected int PageVueNFois
{
    get{
            // Si la variable n'existe pas encore
        if(ViewState["PageVueNFois"] == null)
        {

            return 0;
        }
        else
        {

            return Convert.ToInt32(ViewState["NumberOfPosts"]);
        }
    }
```

Comment ajouter et utiliser une variable globale
```
        set{
            ViewState["PageVueNFois"] = value;
        }
    }
```

ASP.NET met à votre disposition beaucoup de collections qui utilisent la même syntaxe basée sur le dictionnaire. Cela concerne les collections que vous utiliserez pour l'état de la session et l'état de l'application, de même pour les collections utilisées pour le caching et les cookies.

II-D-4 - Désactiver le View State

Par défaut le view state est activé dans une application ASP.NET. La taille de l'information stockée dans le view state augment la taille de l'HTML d'une page Web. ASP.NET met à disposition une flexibilité complète pour désactiver le view state à plusieurs niveaux :

II-D-5 - La protection du View State

Les informations du view state sont stockées dans un string. Mais le string n'est pas encrypté. Les informations du viewstate sont simplement rassemblées ensemble en mémoire puis converties en un string encodé avec le format base64. Vos données ne sont donc pas sécurisées dans le view state.

Pour rendre votre données dans le view state plus sécurisées, vous avez deux options.

ASP.NET met à votre disposition un mécanisme pour savoir si une personne a modifié le contenu du view state pour corrompre votre application. Dans cette technique le view state est encodé en utilisant le hash code (avec l'algorithme SHA1 ou MD5) lorsqu'il est envoyé vers l'exploreur. Lorsque la page est renvoyée avec un post back, ASP.NET vérifie le view state encodé pour voir s'il n'a pas été modifié par le client. Ce type de vérification est appelé machine authentication check (MAC, Vérification d'authentification de la machine).

Un code hash est parfois décrit comme un checksum fort (somme de vérification) en cryptographie. L'idée derrière est qu'ASP.NET examine toutes les données dans votre view state et les font passer à traver un algorithme de hashage avec l'aide d'une valeur secrète de la clé. Cet algorithme de hashage crée un court segment de donnée qui est le code hash. Ce code est alors ajouté à la fin des données du view state. Lorsque la page est renvoyée par postback, ASP.NET examine les données du view state et recalcule le code hash en utilisant le même processus. Puis il contrôle que le checksum qu'il a calculé correspond au code hash qui est stocké dans le view state pour la page. Si un petit malin a changé une partie des données du view state, ASP.NET va finir avec un nouveau code hash qui ne correspond pas. ASP.NET rejetterait alors le postback totalement.

Par défaut, ASP.NET possède l'entrée suivante dans le fichier machine.config :

MAC dans le fichier machine.config
```
   <pages EnableViewStateMac="true" />
```

Cette option permet aux applications tournant sur le serveur Web de se protéger des données modifiées. Vous pouvez également activer ou désactiver la vérification de mofication de données au niveau de la page en initialisation l'attribut EnableViewStateMac de la directive Page à true ou false dans la page ASPX :

MAC désactivé
```
<%@ Page EnableViewStateMac="false" /%>
```

Par occasion, les développeurs peuvent vouloir désactiver cette fonctionnalité pour éviter les problèmes dans une ferme web (web farm) où les serveurs utilisent des clés différentes. En effet un problème peut survenir si la page est renvoyée par post back mais est gérée par un nouveau serveur, qui ne serait pas en mesure de vérifier les informations du view state. Pour désactiver les codes hash, vous pouvez utiliser l'attribut enableViewStateMac de l'élément <pages> dans le fichier web.config ou machine.config. Bien entendu la meilleure solution est de configurer plusieurs serveurs pour qu'ils utilisent la même clé.

Cependant cette option ne vous protège que de la modification du view state par le client, cela n'empêche pas aux utilisateurs de déterminer le contenu du view state. Bien que les valeurs ne soient pas directement lisibles comme avec les query strings ou variables cachées, des utilisateurs déterminées peuvent facilement décoder le view state.

Vous pouvez activer l'encryptage pour une page individuelle en utilisant la propriété ViewStateEncryptionMode de la directive Page

```
<Page ViewStateEncryptionMode="Always">
```

Ou vous pouvez utiliser le même attribut dans un fichier de configuration:

```
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
<system.web>
        <pages viewStateEncryptionMode="Always"/>
</system.web>
</configuration>
```

Vous avez la possibilité entre 3 paramètrages pour l'encryptage du view state. Always (encrypte toujours), Never (ne jamais le faire, Auto (faire l'encryptage seulement si un contrôle le demande spécifiquement. La valeur par défaut est Auto, cela signifie qu'un contrôle doit faire appel à la méthode Page.RegisterRequiresViewStateEncryption() pour demander un encryptage. Si aucun contrôle ne fait appel à cette méthode pour indiquer qu'il contient une information importante à sécuriser, le viewstate n'est pas encrypté. Bien entendu si le mode d'encryption est "Never", l'appel de la méthode Page.RegisterRequiresViewStateEncryption() par le contrôle n'aura aucun effet.

Avec juste quelques changements de configuration, vous pouvez demander à ASP.NET d'encrypter le contenu du view state en utilisant l'algorithme symétrique Triple DES (3DES), ce qui rends le décodage très difficile pour les clients. Ce genre d'encryption peut n'être appliqué qu'aun niveau de la machine en précisant le paramétrage suivant dans le fichier machine.config :

Encryption du View State
```
        <machineKey validation='3DES' />
```

Lors de la sécurisation du view state pour les applications tournant sur une configuration de Web farm, vous devez utiliser la même clé de validation sur toutes les machines de la ferme. Vous devez utiliser une clé assignée manuellement au lieu de la clé générée automatiquement par défaut avec le paramètre <machineKey> dans le fichier machine.config.

II-D-6 - Comment stocker un objet personnalisé dans le view state

II-E - Que choisir comme technique pour gérer l'état du côté client?

Le tableau suivant vous montre les avantages et inconvénients des différentes techniques de gestion de l'état du côté client. Cela vous aidera à prendre rapidement une décision sur choisir quelle technique de gestion de l'état du côté client:

Techniques	Avantages	Inconvénients
Query String	Ne nécessite aucune opération postback	Beaucoup d'exploreurs limite la longueur de la donnée qui peut être inclue dans le query string Aucune sécurité Aucune option de persistence Aucun support pour les valeurs structurées
Les cookies	Ne nécessite aucune opération postback L'état peut être conservé sur l'ordinateur de l'utilisateur	Des utilisateurs désactivent les cookies dans leur exploreur Restriction de la taille Aucun support pour le stockage des valeurs structurées Aucune sécurité
Les champs cachés	Peuvent être utilisés pour des pages qui se postent sur elles même ou vers d'autres pages	Augmentent la taille HTML. Aucun support pour les valeurs structurées Aucune sécurité Aucune option de persistance
View state	Support pour les valeurs structurées Demande moins de codage Options de configuration pour la sécurité	Augmente la taille HTML Ne fonctionne que lorsque la page se poste sur elle même avec un post back Aucune option de persistance

III - Techniques du côté serveur pour la gestion de l'état

ASP.NET prends en charge la gestion de l'état du côté serveur à deux niveaux. Au niveau de l'application Web en utilisation l'état de l'application (application State) et au niveau de la session de l'utilisateur en utilisant l'état de la session (Session State).

III-A - L'état de la session (Session State)

Une application ASP.NET crée une session pour chacun des utilisateurs qui envoit une requête vers l'application. ASP.NET distingue chacune des sessions en envoyant un unique SessionID vers l'explorateur faisant la requête. Ce SessionID est envoyé en tant que cookie ou est imbriqué dans l'URL, cela dépends de la configuration de l'application.

Ce mécanisme d'envoyer le SessionID permet de s'assurer que lorsque la prochaine requête est envoyé vers le serveur, le serveur peut se servir de cet unique SessionID pour identifier de manière unique l'utilisateur qui fait une nouvelle requête vers le même site. Ceci tant que la session soit encore en cours.

Cette possibilité d'identifier de manière unique les requêtes peut être exploité par les développeurs Web pour stocker des données spécifique à une session. Un exemple courant est de stocker le contenu d'un chariot des utilisateurs lorsqu'ils explorent le site. Ces informations spécifiques à une session représentent l'état de la session d'une application Web.

III-A-1 - Comparaison de l'état de la session ASP.NET avec celui d'ASP

ASP.NET possède une nouvelle implémentation de l'état de la session qui supprime tous les problèmes associés avec ASP. Le tableau suivant vous montre les améliorations:

ASP	ASP.NET
ASP garde l'état dans le même processus que celui qui sert d'hôte à ASP. Si jamais le processus ASP pour une raison ou une autre plante, l'état session sera aussi perdu	ASP.NET vous autorise à stocker l'état de la session en dehors du processus dans un state service (service gérant l'état) ou dans une base de données (comme SQL Serveur)
Chaque serveur web ASP maintient son état de session state. Cela peut engendrer un problème avec un Web farm où la requête d'un utilisateur peut être dynamiquement dirigée par un routeur vers d'autres serveurs dans le Web farm	Parce que ASP.NET peut stocker son état de session en dehors du processus plusieurs ordinateurs d'un web farm peuvent utiliser un ordinateur en commun comme serveur prennant en charge l'état de la session
Les sessions ASP ne fonctionnent pas avec les exploreurs ne prennant pas en charge les cookies ou lorsque les utilisateurs ont désactivé les cookies.	ASP.NET prends en charge les sessions sans cookies en stockant le SessionID à l'intérieur de l'URL même en changeant la configuration de l'application

III-A-2 - Comment utiliser l'état de la session

ASP.NET utilise une instance de la classe HttpSessionState pour fournir un accès aux données de la session pour l'utilisateur qui a lancé la requête. Dans une page ASPX, cet object est accessible par l'intermédiaire de la propriété Session de la classe Page. Cette propriété donne accès à l'object HttpSessionState qui stocke l'état de la session sous la forme d'une collection de paires clé-valeur où la clé est du type string et la valeur peut être de n'importe quel type qui est dérivé de System.Object. Je vais exposer les différentes propriétés et méthodes de la

classe HttpSessionState classe:

Propriétés de la classe HttpSessionState

Propriété	Description
CodePage	Spécifie l'identificateur codepage de la session en cours. Cette propriété est présente pour rester compatible avec ASP. Vous devriez utiliser Response.ContentEncoding.CodePage à la place.
Contents	Obtient une référence à l'objet représentant l'état de la session (HttpSessionState). Cela est présent pour rester compatible avec ASP
Count	Permet d'obtenir le nombre d'objets dans l'état de la session
IsCookieless	Indique si la session est gérée en utilisant une session sans cookies
IsNewSession	Indique si la session a été juste crée avec la requête en cours
IsReadOnly	Indique si la session est en lecture seule ou pas
IsSynchronized	Indique si la collection de valeurs d'état de session est synchronisé (thread-safe)
Keys	Permet d'obtenir une collection de tous les clés de la session
LCID	Specifie l'identificateur de paramètres régionaux (LCID) de la session en cours.
Mode	Permet de paraméter le mode de l'état de la session en cours. Les valeurs sont définies par l'énumération SessionStateMode : Off (désactivé), InProc (valeur par défaut, l'état de la session est stocké dans le même processus que aspnet_wp.exe), SqlServer (l'état de la session est stocké dans SQL Server), and StateServer (l'état de la session est stocké dans le service d'état: state service).
SessionID	Représente l'unique identificateur session pour identifier une session.
StaticObjects	Retourne un objet HttpStaticObjectsCollection qui contient les objets déclarés dans les balises <object runat="server" scope="Session"> dans le fichier d'application ASP.NET global.asax
SyncRoot	Retourne un objet qui peut être utilisé pour synchroniser l'accès à la collection des valeurs de l'état de la session
Timeout	Représente le timeout en minutes autorisé entre les requêtes avant que le fournisseur d'état de session met fin à la session

Méthodes de la classe HttpSessionState

Méthode	Description
Abandon()	Annule la session en cours
Add()	Ajoute un objet identifié par son nom à l'état de la

Méthode	Description
	session
Clear()	Supprime tous les objets de l'état de la session
CopyTo()	Copie les valeurs de l'état de la session vers un tableau à une dimension à l'index spécifié
GetEnumerator()	Retourne un objet énumérateur pour les valeurs de l'état de la session dans la session en cours
Remove()	Vire un objet donné de l'état de la session
RemoveAll()	Supprime tous les objets de l'état de la session. Elle fait appel à la méthode Clear() à l'intérieur de son code
RemoveAt()	Vire un objet donné de l'état de la session à un index donné

Le code suivant vous montre comment utiliser la session pour récupérer un valeur:

Utilisation de la session
```
private void Page_Load(object sender, System.EventArgs e)
{
    if(Session["Utilisateur"] == null)
    {
        // La clé Utilisateur n'est pas présente dans l'état de la session

    }
    else
    {
        // La clé Utilisateur est présente dans l'état de la session
        lblUtilisateur.Text="Bienvenue chez nous, " + Session["Utilisateur"].ToString();
    }
...
}

private void btnSubmit_Click(object sender, System.EventArgs e)
{
    // Ajoute le nom de l'utilisateur dans la session
    Session["Utilisateur"] = txtUtilisateur.Text;

}
```

L'état de la session n'est pas stocké de manière longue comme les cookies. La technique utilisée par défaut pour passer le SessionID est l'utilisation de cookies qui ne persistent pas. A savoir si vous fermer l'exploreur et que vous relancez la requête, la session d'auparavant sera perdue.

Pour utiliser la session sans cookies, vous devez paramétrer votre fichier web.config associé avec votre application en mettant l'attribut cookieless à true dans l'élement XML <sessionState> commme ci-dessous:

Paramètrer le fichier web.config pour utiliser la technique sans cookies pour gérer l'état de la session
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.web>
        <sessionState mode="Inproc"
              cookieless="true" />

    </system.web>
</configuration>
```

III-B - L'état de l'application (Application State)

L'état de l'application est utilisé pour stocker les données qui sont utilisées globalement à travers toute l'application. L'état de l'application est stockée en mémoire et contrairement à l'état de la session, l'état de la session ne peut pas être configuré pour être stocké sur un autre serveur ou une base de données SQL. Cela limite l'utilité de l'état

de l'application dans le cas de web farm.

L'état de l'application peut être facilement accèdé par la propriété Application de la classe Page. Cette propriété fournit un accès à l'objet HttpApplicationState qui stocke l'état de l'application en tant que collection de paires clé-valeur où la clé est du type string et la valeur peut être de tout type dérivé de System.Object. Les tableaux suivant vous présente les propriétés et méthodes de la classe HttpApplicationState class:

Propriétés de la classe HttpApplicationState

Propriétés	Description
AllKeys	Retourne la collection de tous les noms des clés de l'état de l'application. Cette collection est retournée comme un tableau de strings
Contents	Permet d'obtenir une référence à l'objet de l'état d'application HttpApplicationState
Count	Détermine le nombre d'objets dans l'état de l'application
Keys	Retourne la collection NameObjectCollectionBase.KeysCollection de tous les noms des clés dans l'état de l'application
StaticObjects	Retourne tous les objets déclarés par une balise <object runat="server" scope="Application"></object> dans l'application ASP.NET.

Méthodes de la classe HttpApplicationState

Méthode	Description
Add()	Ajoute un objet à l'état de l'application
Clear()	Supprime tous les objets de l'état de l'application
Get()	Retourne un objet de l'état de l'application par le nom de la clé ou son index.
GetKey()	Retourne un objet de l'état de l'application par son index
Lock()	Bloque l'accès à un objet de l'état de l'application. Cela est utilisé pour éviter que d'autres clients changent les données stockées dans l'état de l'application
Remove()	Supprime tous les objets de l'état de la session. Elle fait appel à la méthode Clear() à l'intérieur de son code
RemoveAt()	Vire un objet donné de l'état de la session à un index donné

Le code suivant vous montre comment utiliser la propriété Application pour stocker des données de portée globale à l'application:

Utilisation de l'état de l'application

```
private void Page_Load(object sender, System.EventArgs e)
{
    // Bloque l'Application avant de modifier l'état de l'application
    Application.Lock();
    if(Application["Visites"] != null)
    {
```

Utilisation de l'état de l'application

```
            // Augmente la variable du nombre de visites dans l'état de l'application
            Application["Visites"] = (int) Application["Visites"] + 1;
        }
        else
        {
            Application["Visites"] = 1;
        }
        // Débloque l'Application lorsque les changements sont terminés
        Application.UnLock();
        //  Récupération de la valeur du nombre de visites à partir de l'état de l'appplication
        lblInfo.Text = "Cette page a été visitée (" +
          Application["Visites"].ToString() + ") fois!";
    }
```

Cette technique pour garder le nombre de visites sur cette page est loin d'être fiable car elle ne stocke que le nombre de fois la page a été visitée depuis que l'application a démarré! Si vous souhaitez conserver ce nombre à travers plusieurs redémarrages de l'application, vous devriez conserver ce nombre dans une base de données de manière périodique.

Vous avez modifié ci-dessus le contenu de l'état de l'application en utilisant les méthodes Application.Lock() et Application.UnLock(). bloquer est important pour pouvoir garder l'état de l'application fiable lorsque plusieurs utilisateurs peuvent modifier le contenu de l'objet application de façon concurrente. Bien que l'application soit bloquée, sur l'utiliseur en cours Ce mécanisme de bloquage peut nuire sérieusement la capacité de monter en charge d'une application Web. En conséquence vous ne devriez pas stocker des données modifiable pour une mise à jour dans l'état de l'application.

Vous n'avez pas besoin d'utiliser les méthodes Application.Lock() et Application.Unlock() dans les gestionnaires d'événerment Application_Start() et Application_End() car ces gestionnaires d'événement ne sont appelés qu'une seule fois pendant la durée de vie d'une application

ASP.NET met à disposition un autre mécanisme pour maintenir l'état global pour une application en utilisant le cache de données de l'application. Ce cache de données de l'application devrait représenter votre premier choix pour stocker les données globales dans une application ASP.NET.

III-C - Utilisation du Caching

Caching représente le stockage des informations afin de pouvoir les récupérer facilement ultérieurement plutôt que de les regénérer à partir de rien chaque fois qu'elles soient demandées. Par exemple une page Web peut être cachée afin qu'elle soit rendue plus rapidement lorsqu'elle est demandée pour la deuxième fois.

En mettant en cache, vous pouvez alléger la charge sur le serveur, car il n'a pas à regénérer le résultat (output) pour chaque requête d'une page. Mais cela signifie que le résultat (output) envoyé vers le client risquerait de n'être pas identique à celui que le client recevrait si aucun cache n'était utilisé. Vous devez donc équilibrer vos besoins de des données courantes et la charge sur le serveur en spécifiant une date d'expiration pour tout contenu mis en cache.

ASP.NET implémente 3 types de caching:

III-C-1 - Cache de la sortie de la page (Output Caching)

Le mise en cache de la page en sortie (Output caching) fait référence à la mise en cache de l'intégralité de la sortie de la requête d'une page.

Vous pouvez préciser la mise en cache d'une page ou d'un contrôle utilisateur en utilisant la directive OutputCache. Le tableau vous présente tous les attributs de cette directive:

Attributs de la directive OutputCache

Attribut	Description
Duration	Précise la durée en secondes pendant laquelle la page ou le contrôle utilisateur resterait mis en cache
Location	Précise l'endroit du cache de la sortie pour une page. Les valeurs possibles sont spécifiées pour l'énumération OutputCacheLocation et comprends la valeur Any (valeur par défaut, la mise en cache peut être mis sur l'exploreur du client, sur le serveur proxy ou sur le server Web), Client, Downstream (l'exploreur du client ou le serveur proxy), None (aucune mise en cache), et Server. Cet attribut n'est pas pris en charge lorsqu'il est appliqué à la directive OutputCache dans les contrôles utilisateur car les contrôles utilisateurs doivent résider sur le serveur Web pour être assemblés
VaryByControl	Précise une liste de contrôles séparés par des points virgule dans le contrôle utilisateur à partir duquel la mise en cache peut varier. Cet attribut n'est pris en chage que pour les contrôles utilisateur.
VaryByCustom	Spécifie un string qui indique soit l'exploreur ou un string personnalisé pour faire varier la mise en cache en sortie. Si la mise en cache doit varier avec un string personnalisé, vous devriez fournir une version surchargée de la méthode HttpApplication.GetVaryByCustomString() dans le fichier global.asax pour indiquer comment la page devrait mise en cache.
VaryByHeader	Spécifie une liste d'en-têtes HTTP séparés par des points virgule HTTP headers qui serviront à faire varier la mise en cache. Cet attribut n'est pas pris en charge lorsqu'il est appliqué à la directive OutputCache dans les contrôles utilisateur.
VaryByParam	Spécifie une liste de paramètres de la page web séparés par des points virgule à partir desquels la mise en cache en sortie peut varier. Les valeurs possibles sont None (La mise en cache ne dépends pas des paramètres), * (Mise en cache pour chaque ensemble distinct de paramètres), toute clé en format string contenue dans une requête GET, ou tout nom de paramètre dans une requête POST

Lorsqu'une directive OutputCache est appliquée à une page ASPX, les deux attributs Duration et VaryByParam doivent être précisés. Dans le cas des contrôles utilisateur l'attribut VaryByParam ou

VaryByControl doit être précisé avec l'attribut Duration. Si les attributs demandés ne sont pas fournis une erreur de compilation sera lancée.

Pour montrer en action la mise en cache de la sortie rendue par une page, nous allons utiliser le code suivant:

Affichage de l'heure en cours
```
private void Page_Load(object sender, System.EventArgs e)
{
    lblTemps.Text = DateTime.Now.ToLongTimeString();
}
```

La page affiche l'heure en cours sur le label. Faites une rafraichissement de la page plusieurs fois et vous pouvez constater que l'heure affichée change chaque fois que vous recharger la page.

Pour conserver l'aspect de la page, on la met en cache. Cela permet de récupérer l'état d'une page plus rapidement. Il ne faut utiliser la mise en cache de la page que si l'état de la page ne varie pas souvent évidement!

Pour mettre en cache la page il suffit d'ajouter la directive suivante :

Directive pour mettre en cache le contenu de la page
```
<%@ OutputCache Duration="15" VaryByParam="None" %>
```

Relancez le projet la feuille affichera l'heure en cours. Faites un rafraichissement plusieurs fois de la page et vous pouvez remarquer que l'heure affichée ne change pas. Attendez 15 secondes après la requête d'origine et vous contasterez que l'heure changera.

*Dans l'exemple ci-dessous, on a utilisé la directive OutputCache, cette directive demande obligatoirement l'utilisation de l'attribut **VaryByParam**. Si la sortie de la page ne dépends d'aucun paramètres en entrée, vous pouvez utiliser la valeur **None** (Aucun paramètre) comme valeur de cet attribut.*

Sinon, précisez le nom du paramètre qui peut causer un changement de la mise en cache.

Si vous souhaitez obtenir plus de contrôle sur la mise en cache de la page en sortie par programmation, cela est possible à l'aide de la classe **HttpCachePolicy** qui vous permet d'initialiser la mise en cache d'une page en sortie à l'aide de votre code.

La propriété Cache de l'objet **HttpResponse** fournit un accès à l'instance de la classe **HttpCachePolicy**. L'objet du type **HttpResponse** peut être récupéré par la propriété **Response** de la classe **Page** ou de la classe

HttpContext.

Le code suivante vous montre comment :

Paramètrage la mise en cache de votre page en sortie par programmation à l'aide de l'objet du type HttpCachePolicy renvoyé par la propriété Cache de la classe HttpResponse

```
private void Page_Load(object sender, System.EventArgs e)
{
   lblTemps.Text = DateTime.Now.ToLongTimeString();
   Response.Cache.SetExpires(DateTime.Now.AddSeconds(15));
   Response.Cache.SetCacheability(HttpCacheability.Public);
   Response.Cache.SetValidUntilExpires(true);
}
```

Le code manipule l'objet HttpCachePolicy exactement de la même façon que lors du paramètrage de la direactive OutputCache de l'exemple ci-dessous. Cet exemple utilise 3 méthodes de l'objet HttpCachePolicy.

La méthode **SetExpires()** spécife un temps d'expiration pour la version de la page mise en cache, il est de 15 secondes à partir du moment la page est générée.

La méthode **SetCacheability()** précise où peut être mise en cache la sortie par l'intermédiaire de l'énumération **HttpCachePolicy** : NoCache pour aucune mise en cache, Private pour une mise en cache sur le client (la valeur par défaut), Public pour une mise en cache sur tout serveur proxy aussi bien que sur le client, et Server pour mettre en cache le document seulement sur le serveur Web.

La méthode **SetValidUntilExpires()** avec son paramètre initialisé à true indique au serveur d'ignorer les tentatives du côté client de rafraîchir le contenu jusqu'à son expiration.

La mise en cache de plusieurs versions d'une page peut être utile chaque fois que vous avez une page dont le contenu dépends d'un paramêtre en entrée. Vous pouvez baser cette mise en cache multi version sur tout attribut HTTP Header ou de l'exploreur, mais plus souvent vous utiliserez l'attribut VaryByParam pour mettre en cache plusieurs valeurs qui dépendent d'une query string (chaîne de caractères passée en url voir plutôt) ou d'un paramêtre passé par un POST Form (passage d'une valeur par la méthode post lors d'un formulaire).

Par exemple si vous avez une page qui est accédée en utilisant une URL comme http://NomServeur/InfosPourCePays.aspx?pays=France et que le résultat de la page en sortie dépends du paramètre string passé dans la requête, vous pouvez mettre en cache les différentes versions de cette page en utilisant la directive **OutputCache** suivante:

directive utilisée pour mettre en cache une page dont la sortie dépends d'un paramètre

```
<%@ OutputCache duration="20" VaryByParam="pays"%>
```

La valeur de l'attribut VaryByParam est le même nom que celui du paramètre passé dans la chaine de caractères utilisés dans le requête (query string) et la durée du cache est 20 minutes. Ce type de mise en cache permet de limiter des aller-retours vers le serveur de base de données. De plus le server Web peut fournir les données à partir du cache en sortie (output cache) sans à avoir à regénérer dynamiquement la page à chaque requête. En conséquence la mise en cache permet de créer des applications web performantes et répondant bien à la montée en charge (scalable).

Vous pouvez aussi mettre en cache la page en sortie avec ce cache dépendant de la valeur d'un contrôle

III-C-2 - Mise en cache de fragments de page (Fragment Caching)

La Mise en cache de fragments de page fait référence à la mise en cache de certaines parties d'une page donnée. Vous pouvez encapsuler une portion d'une page dans une contrôle utilisateur et mettre en cache cette portion de page tout en demandant le reste de la page d'être généré de manière dynamique lors de chaque requête.

La Mise en cache de fragments est similaire à la mise en cache de la sortie, où dans ce cas on met en cache la sortie générée par un contrôle utilisateur. La directive **OutputCache** met aussi en cache les contrôles utilisateur. Revoyez le tableau ci-dessus pour cette directive pour vous rafraichir la mémoire :-).

Vous devriez spécifier soit l'attribut VaryByParam ou l'attribut VaryByControl dans le directive OutputCache du contrôle utilisateur. Vous devriez aussi préciser l'attribut Duration pour indiquer la durée de la mise en cache pour ce contrôle utilisateur. Un nombre quelconque de contrôles utilisateurs peuvent exister dans une page et chacun de ces contrôles peuvent maintenir leur mise en cache.

Lors de la mise en cache des contrôles utilisateurs, vous devriez considérer les points suivants :

- Les contrôles utilisateurs mis en cache ne sont pas accessibles par programmation, vous obtiendriez une exception si vous essayiez
- Les contrôles utilisateurs mis en cache ne sont pas ajoutés à l'arborescence de contrôles de la page
- Les contrôles utilisateurs mis en cache ne peuvent pas utiliser la liaison de données. Si vous essayez d'effectuer une liasion de données sur ces contrôles, une exception surviendrait

III-C-3 - La mise en cache des données

ASP.NET vous permet aussi de mettre en cache les données de votre application. Dans une application Web, les données peuvent être récupérées par l'intermédiaire d'opérations coûteuses et prennant du temps. Une grande majorité de ces données ne changent pas souvent mais sont demandées fréquemment.

La mise en cache de données arbitraires revient à faire une mise en cache de données ou une mise en cache de données de l'application.

La classe **Cache** de l'espace de nom **System.Web.Caching** vous permet d'insérer tout objet dans le cache de données en donnant une clé et l'objet à mettre en cache. Vous pouvez ensuite récupérer cet objet par programmation en fournissant cette clé. Vous pouvez mettre en cache des simples strings, des listes de tableaux, des datasets, des hash tables et même des objets personnalisés!

Vous pouvez préciser un temps donné d'expiration ou un temps d'expiration variable pour les données mises en cache.

La mise en cache des données vous permet de préciser toute dépendance pour l'élement mis en cache. Par exemple vous pouvez mettre en cache une chaîne de connexion en demandant la recréation du cache si jamais un fichier XML contenant cette chaine de connexion est modifié. Si vous paramètrez bien les dépendances, vous pouvez garantir que les utilisateurs obtiendront des données mises à jour comme vous le souhaitez!

Vous pouvez paramètrer la priorité de votre élément mis en cache en utilisant l'énumération **CacheItemPriority** avec comme valeurs AboveNormal, BelowNormal, Default (Normal), High, Low, Normal, NotRemovable. En paramètrant la priorité, vous pouvez être sûr que les données (récupérées par l'intermédiaire d'une opération couteuse) ne seront pas virées et ces données seront facilement récupérables et les données facilement récupérables ou crées sont enlevées du cache au cas où la mémoire commence à manquer sur le serveur.

Lorsque la mémoire commence à diminuer grave, ASP.NET est en mesure d'enlever les données du cache si les données dans le cache ne sont pas utilisées fréquemment ou ne sont pas importantes. Cette opération est appelée **scavenging** en anglais, pas d'équivalent en français mais cela signifie passer en revue les données du cache pour se débarasser des données considérées comme pas critiques. Ce processus scavenging permet d'éviter le recyclement du processus ASP.NET worker chaque fois que la mémoire est trop limite pour être en mesure de contenir des nouvelles requêtes.

Vous pouvez également fournir une méthode de rappel (callback method) qui est appelée chaque fois qu'un élément mis en cache est enlevé du cache. L'élément de donnée peut être enlevé du cache pour plusieurs raisons qui sont explicitées dans l'énumération **CacheItemRemoveReason** y compris **DependencyChanged**,**Expired**,**Removed** (par la méthode Cache.Remove()) et **UnderUsed** (enlevé lorsque la mémoire est basse). La méthode de rappel est averti qu'un élément du cache est enlevé et selon la donnée mise dans le cache, cette méthode peut être appelée pour effectuer un nettoyage, une trace de log, une mise à jour ou une récréation de la donnée.

L'objet **Cache** peut être facilement accéder en utilisant la propriété **Cache** de l'objet **Page** ou de l'objet **HttpContext**. Vous pouvez insérer ou ajouter des éléments au cache de données en utilisant la méthode **Insert()** ou **Add()** de l'objet **Cache** respectivement. La méthode **Insert()** possède 4 versions surchargées qui fournissent le choix sur comment insérer les éléments dans le cache en spécifiant des paramètres tels que le temps d'expiration, le temps d'expiration variable, la dépendance du cache, la priorité du cache et la méthode de rappel. La méthode **Add()** ajoute un élément au cache de données et retourne une référence à l'élément ajouté. La méthode **Remove()** est appelée pour enlever explicitement des éléments du cache de données.

Vous pouvez aussi facilement insérer des données dans le cache de données en utilisants les indexeurs en fournissant une clé et une valeur comme dans l'exemple ci-dessous:

Insérer un objet dataset dans le cache de données de l'application à l'aide de l'indexeur

```
Cache["DataSetClients"] = dsClients;
```

IV - Les avantages et inconvénients de toutes les méthodes de gestion de l'état dans une application ASP.NET

ASP.NET met à notre disposition une très grande variété d'options différentes pour la gestion de l'état. On y trouve notamment les collections de l'état de la session et de l'application comme avec ASP (mais avec quelques améliorations) et un tout nouveau modèle le view state. ASP.NET comprends de plus un système de mise en cache qui vous permet de garder les informations sans pour autant sacrifier la capacité à monter en charge de l'application (scability). Chaque choix de gestion de l'état possède une espérance de vie, une portée, une performance et un niveau du support différent.

	View State	Query String	Cookies personnalisés
Types de données autorisés	Tout type de donnée .NET sérialisable	Des Strings de longueur limitée	Des Strings
Localisation du stockage	Dans un champs caché contenu dans la page web en cours	Dans le String URL de l'exploreur	Dans l'ordinateur du client (en mémoire ou dans un fichier texte)
Durée de vie	Données conservées de manière permanente lors du postback vers une même page	Données perdues lorsque l'utilisateur entre une nouvelle URL ou lorsqu'il ferme l'exploreur.	Durée déterminée par le développeur. Ils peuvent être utilisés à travers plusieurs pages et peuvent persister entre plusieurs visites
Portée	Limitée à la page en cours	Limitée à la page cible	Disponible à travers toute l'application ASP.NET
Sécurité	Par défaut cette méthode n'est pas sure. Mais vous pouvez utiliser les directives de la page pour imposer un encryptage ou un hashage	Données facilement visualisables et faciles à modifier pour l'utilisateur (à moins de les encrypter)	Données non sures et facilement modifiables (à moins de les encrypter)
Implications sur la performance	Stocker une quantitée importante d'informations peut ralentir la transmission	Aucune incidence	Aucune incidence
Usage courant	Utilisé lors du paramètrage de la page	Permet d'envoyer par exemple l'ID d'un produit	Permet de personnaliser les préférences d'un site web

	Etat de la Session	Etat de l'Application	Mise en Cache ASP.NET (Caching)
Types de données autorisés	Tout type de donnée .NET sérialisable. Les types de données non sérialisables sont pris en charge si vous utilisez le service d'état in-process par défaut	Tout type de donnée	Tout type de donnée
Localisation du stockage	Mémoire du serveur	Mémoire du serveur	Mémoire du serveur
Durée de vie	Expire après une période prédéfinie (habituellement 20 minutes mais cela peut être changé globalement ou par programmation)	Durée de vie de l'application	Durée de vie dépends de la politique d'expiration que vous utilisez (mais peut être libéré prématurement si la mémoire commence à

	Etat de la Session	Etat de l'Application	Mise en Cache ASP.NET (Caching)
			manquer)
Portée	Toute l'application ASP.NET	Toute l'application ASP.NET	Toute l'application ASP.NET
Sécurité	Très sure car les données ne sont jamais transmises au client	idem	idem
Implications sur la performance	Stocker une quantitée importante d'informations peut sérieusement ralentir le serveur surtout quand beaucoup d'utilisateurs sont connectés en même temps	Stocker une quantitée importante d'informations peut sérieusement ralentir le serveur car ces données n'expirent jamais et ne seront jamais enlevées	Stocker une quantitée importante d'informations peut forcer la suppression d'autres informations plus importantes mis en cache. Cependant ASP.NET a la possibilité d'enlever des éléments prématurement pour s'assurer d'obtenir une performance optimale
Usage courant	Stockage d'informations sur l'utilisateur (Panier lors d'un e shopping)	Stockage de tout type de donnée	Stockage de données récupérées à partir de la page de données (DataSet)