Tutoriels Asp

par Florian (homepage)

Date de publication : 14/02/2003

Dernière mise à jour : 26/03/2007

Ce tutoriel est destiné à une prise en main rapide de l'Asp.

I - Introduction

Le but de ce petit tutoriel est de vous "mettre le pied à l'étrier", pour pouvoir utiliser ASP rapidement.

Il n'a pas la prétention d'être exhaustif.

A l'origine je l'ai écrit début 2003, disposant à l'époque d'un serveur Windows NT, donc d'Asp 2.0. Depuis, je dispose de serveurs W2K3, et je suis passé directement à Asp .Net.

Pour cette raison, ce tutoriel n'abordera pas Asp 3.0 .

Les exemples donnés sont fonctionnels et destinés à être adaptés par vous pour vos besoins.

Ils sont tous écrits en VBScript.

Il est supposé que vous avez des notions de HTML et de VBScript suffisantes.

II - Petite Faq

II-A - L'ASP, à quoi ça sert?

ASP (Active Server Pages) sert à créer des pages dynamiques, c'est à dire des pages dont le contenu pourra être différent à chaque accès, en fonction de la personne qui consulte, de la date, du contenu d'une base de données, etc.

II-B - Comment je me procure les logiciels nécessaires?

A partir de Windows 2000 Serveur, Internet Information Server est inclus, donc rien à rajouter.

Pour NT4 : Le "moteur" ASP 2.0 est fourni avec Internet Information Server 4, par le biais de l'Option Pack. L'option pack est disponible gratuitement a cette adresse:

Option Pack pour NT

Il est possible d'utiliser ASP avec Personal Web Server sur les postes de travail Windows "non serveurs" (Windows 98,Windows XP). Les versions "familiales" des dernières moutures de Windows ne permettent pas l'éxécution de Internet Information Server, et donc l'usage de ASP.

Pour utiliser ASP sur d'autres plateformes, il existe le produit Chili!Soft ASP (appelé désormais "Sun ONE Active Server").

II-C - Comment dois-je faire pour créer des pages ASP?

Une page ASP n'est rien d'autre qu'un fichier texte avec l'extension .asp. C'est un mélange de HTML et de code ASP. Un bon éditeur de texte est suffisant, bien que de nombreux outils puissants existent pour assister le développeur pour la mise en page et la syntaxe.

Pour tester vos pages, un serveur opérationnel et un répertoire dans lequel vous aurez les droits d'écriture sont suffisants.

Les commandes ASP peuvent être rédigées en Javascript ou VBScript.

Tous les exemples ont été testés sur W2K3/IIS6 et écrit avec un éditeur de texte basique en utilisant VBScript.

II-D - Qu'est-ce qu'on peut faire avec ASP?

Avec ASP, on peut (dans le désordre):

- créer des interfaces via des formulaires (listes déroulantes, cases à cocher, etc.) qui seront créées et modifiés dynamiquement
- accéder à des bases de données via ODBC (Open DataBase Connectivity)
- gérer les cookies
- accéder à des fichiers, des dossiers, les lire, les modifier, les créer
- en utilisant des composants, envoyer des courriers électroniques, faire des graphiques, etc.

Et beaucoup d'autres choses...

II-E - Qu'est-ce qu'on ne peut pas faire avec ASP?

- on ne peut pas *directement* créer des graphiques dans une page ASP
- on ne peut pas modifier la page sans que le serveur la réexamine (Javascript ou Java peuvent le faire)
- on ne peut pas envoyer de courrier électronique (un composant est nécessaire)

II-F - ASP n'est pas PHP...

PHP fonctionne sur plusieurs plateformes, fournit un support intégré pour Mysql ainsi que ne nombreuses autres bases de données; ASP ne tourne "nativement" que sous Win32.

Il est courant d'entendre dire que ASP consomme plus de ressources et est plus malaisé à appréhender que PHP, mais que PHP a des lacunes par rapport à ASP (variables de sessions par exemple).

Il est souvent dit que l'usage de VBScript dans ASP le rend plus accessible que PHP.

Faites vous vous-même votre idée, PHP peut tourner sur le même serveur que celui que vous utiliserez pour ASP.

Le site officiel de Php

III - Notions de base

III-A - Comment ça fonctionne?

La page ASP va être analysée par le serveur, et la page qui sera renvoyée au navigateur sera construite en fonction du code que vous aurez intégré. Une fois la page chargée, si vous examinez le source vous ne trouverez aucune trace de code VBScript, ce ne sera que du code HTML.

On utilise intensivement des "objets", qui possèdent des "collections" et/ou des "propriétés" et/ou des "méthodes", ce qui est assez analogue à de la programmation orientée objet. En ASP 2.0, les objets "basiques" sont : Request,Response,Application,Session,Server.

Il y a aussi les objets dits "de script" : FileSystemObject,Dictionary,Textstream.

Le but des tutoriels n'est pas de rentrer dans les détails, mais de se servir au bon moment des objets utiles.

III-B - Une page ASP simple

Le code dans une page ASP est délimité par:

```
<%
```

en début de code et par :

```
%>
```

en fin de code.

Exemple de page simple, qui va afficher trois blocs de cinq lignes numérotées de 1 à 5:

Page Asp simple
```
<html>
<head>
<title>Ma page ASP</title></head>
<body>
Une petite boucle:<br>
<% for i=1 to 5 %>
Je suis la ligne <%=i%><br>
<% next %><br>
Encore une autre:<br>
<% for i=1 to 5 %>
Je suis encore la ligne <%response.write(i)%><br>
<% next %><br>
Et pour finir:<br>
<% for i=1 to 5
texte = "Je suis toujours la ligne " & i & "<br>"
response.write(texte)
next %>
</body></html>
```

qui donne en résultat:

Résultat de la page Asp simple
```
Une petite boucle:
Je suis la ligne 1
Je suis la ligne 2
Je suis la ligne 3
Je suis la ligne 4
Je suis la ligne 5

Encore une autre:
Je suis encore la ligne 1
Je suis encore la ligne 2
Je suis encore la ligne 3
Je suis encore la ligne 4
Je suis encore la ligne 5

Et pour finir:
Je suis toujours la ligne 1
Je suis toujours la ligne 2
Je suis toujours la ligne 3
Je suis toujours la ligne 4
Je suis toujours la ligne 5
```

L'examen du source donne ceci:

Source d'une page Asp après interprétation par le serveur
```
<HTML><HEAD><TITLE>Ma page ASP</TITLE><BODY>
Une petite boucle:<BR>Je suis la ligne 1<BR>Je suis la ligne 2<BR>Je suis la ligne 3<BR>
Je suis la ligne 4<BR>Je suis la ligne 5<BR><BR>Encore une autre:<BR>Je suis encore la ligne 1<BR>
Je suis encore la ligne 2<BR>Je suis encore la ligne 3<BR>Je suis encore la ligne 4<BR>
Je suis encore la ligne 5<BR><BR>Et pour finir:<BR>Je suis toujours la ligne 1<BR>
Je suis toujours la ligne 2<BR>Je suis toujours la ligne 3<BR>Je suis toujours la ligne 4<BR>
Je suis toujours la ligne 5<BR></BODY></HTML>
```

Ce qui permet de constater que le code ASP a "disparu".

Quelques explications:

la syntaxe "<%= texte %>" permet d'écrire dans la page, elle est équivalente à "<%response.write(texte)%>" ou encore "<%response.write texte%>". C'est la méthode "write" de l'objet "response".

Les deux premières boucles sont donc identiques, la syntaxe "compacte" fournissant juste un confort d'écriture.

Dans la troisième boucle, on fabrique une chaine de caractère en utilisant l'opérateur de concaténation "&". On écrit ensuite le texte dans la page.

III-C - Note

Chaque passage du HTML à l'ASP (c'est à dire chaque fois qu'on inclut du code dans du source HTML) prends du temps au serveur.

Il est rentable d'éviter des "commutations" incessantes de HTML à ASP dans le cas où un grand nombre d'informations sont à écrire (exemple: pour un tableau très volumineux, composer chaque ligne dans une chaine et écrire la ligne directement avec response.write plutôt que d'écrire le code du tableau en HTML et de mettre de l'ASP à l'intérieur des cellules).

Attention, la syntaxe "compacte" (<%= ... %>) de response.write n'est valide que dans le cas où c'est la seule instruction qui se trouve entre le bloc <% ... %>, si ce n'est pas le cas, il faut utiliser la forme "longue" response.write .

IV - Variables de session, d'application

IV-A - Mais qu'est ce que c'est que ça?

Les variables d'application sont initialisées au démarrage du site, et persistent jusqu'à son extinction.

Elles sont visibles et modifiables par toutes les pages des utilisateurs.

Les variables de session sont initialisées à chaque connexion et pour chaque visiteur, et persistent jusqu'à la fin de la session.

Les variables d'application peuvent permettre de conserver le nombre de personnes connectées, le dernier numéro de commande passé sur le site, etc.

Les variables de session sont très utiles, elles permettent de conserver les choix de l'utilisateur, savoir s'il est authentifié ou non, toutes sortes d'informations qui pourront être consultées et modifiées à chaque page.

Elles sont indispensables pour écrire des pages flexibles et efficaces.

Néanmoins, les variables de session occupant de l'espace sur le serveur, ne devraient pas être utilisées pour stocker de grosses quantités de données.

IV-B - Comment initialiser et lire ces variables

Affecter une variable de session:

Session("variable")=valeur

Affecter une variable d'application:

Application("variable")=valeur

Récupérer une variable de session:

valeur=Session("variable")

Récupérer une variable d'application:

valeur=Application("variable")

IV-C - Où et quand initialiser ces variables

A chaque arrêt/démarrage de session ou d'application, un fichier nommé **global.asa** est utilisé pour savoir quoi faire, et entre autres initialiser les variables de session et d'application.

Voici un exemple de fichier:

Exemple de fichier global.asa

```
<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Application_OnStart
Application("connectes")=0
END SUB
</SCRIPT>

<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Application_OnEnd
END SUB
</SCRIPT>

<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Session_OnStart
session("loginok")=0
Application("connectes")=Application("connectes")+1
END SUB
</SCRIPT>

<SCRIPT LANGUAGE=VBScript RUNAT=Server>
SUB Session_OnEnd
Application("connectes")=Application("connectes")-1
END SUB
</SCRIPT>
```

A quel moment les sections sont éxécutées:

Application_OnStart : démarrage de l'application (ou modification du fichier global.asa)

Application_OnEnd : arrêt de l'application

Session_OnStart : démarrage d'une session utilisateur

Session_OnEnd : fin de session utilisateur

Dans cet exemple:

- au démarrage de l'application on initialise un compteur de connectés
- à chaque début de session on augmente de 1 le nombre de connectés et on initialise une variable pour savoir que la personne n'est pas authentifiée (attention c'est juste un exemple, il faut exploiter cette variable pour que ça fonctionne)
- à chaque fin de session on diminue de 1 le nombre de connectés

La fermeture de la session provoque la disparition des variables liées à celle-ci.

Note sur Session_OnEnd

La partie Session_OnEnd n'est pas très fiable, j'ai essayé avec acharnement et sans succès de m'en servir pour modifier une base de données, mais sans résultats.

IV-D - Evitez de modifier les variables d'application de manière anarchique

Il existe une possibilité d'interdire aux autres sessions de modifier les variables d'application, afin de préserver leurs cohérences.

La méthode **Application.Lock** empêchera les autres sessions de modifier les variables d'application.

Ce bloquage persistera jusqu'à l'appel de la méthode Application.Unlock .

Exemple:

Fragment de code utilisant le verrouillage
```
<% Application.Lock
Application("variablesensible") = Application("variablesensible") + 1
Application.Unlock %>
```

V - Utiliser les formulaires

V-A - But et utilité

Un formulaire est un ensemble de champs de saisie, situés à l'intérieur de balises <form> et </form>.

Il a un nom, une méthode d'éxécution et une page "cible" à laquelle il est soumis.

Par soumission on entend le fait que l'état des champs de saisie du formulaire est transmis au serveur pour pouvoir être exploité.

Le but est de pouvoir récupérer des choix utilisateurs, l'utilité est qu'on peut fabriquer des interfaces simples (d'aucun diront simplistes...) pour interagir avec l'utilisateur.

Attention la richesse est loin d'être celle d'une application réalisée par un outil de développement "classique" (C++ Builder ou Delphi par exemple).

Le programmeur est responsable du réaffichage des valeurs dans les éléments du formulaire. Ce besoin disparait dans Asp .Net avec l'"Auto postback".

V-B - Elements disponibles

Le "jeu de base" des éléments de saisie est:

- champ de saisie
- case à cocher
- boutons radios
- liste déroulantes
- champ de saisie de nom de fichier
- champ "zone de texte"
- champ caché
- bouton
- boutons "images"

Cliquer sur **ce lien** pour afficher une page html d'exemple.

Si vous modifiez les valeurs, et que vous cliquez sur le bouton "Envoyer" (ce que vous avez déjà fait j'en suis sur) vous remarquerez qu'aucune de vos modifications n'a persisté. C'est là que l'ASP intervient, en vous permettant de récupérer les choix, les analyser, les contrôler et en repositionnant les valeurs du formulaire.

V-C - Récupérer les données

Tout d'abord, il existe deux façons d'envoyer les données, soit à travers l'URL de la page, soit en "cachant" les données dans un bloc transmis en entête.

La méthode "post" transmet les données en les "cachant", la méthode "get" à travers l'URL (vous pouvez voir les paramètres dans la ligne d'adresse de votre navigateur).

Si vous ne savez pas quoi choisir, utilisez "post". La différence entre les deux méthodes, pour la partie ASP, se situe uniquement dans la méthode de récupération des valeurs. De plus, la longueur des URL n'étant pas illimitée, vous risqueriez d'avoir des problèmes avec des informations trop nombreuses.

Nous allons utiliser l'objet "Request", qui permet d'accéder aux données soumises au serveur (ainsi qu'à bien d'autres choses).

Pour accéder "facimement" aux données du formulaire, il est préférable que chaque élément ait un nom unique.

On accède à un élément nommé "elem" avec la syntaxe suivante (syntaxe pour la méthode "post"):

```
valeur_de_l_element = Request.Form("elem")
```

Si notre élément de formulaire comporte plusieurs "sous-composants" (cas typique: les listes de sélection), on accède au n-ième élément de cette façon:

```
valeur_sous_element = Request.Form("elem")(n)
```

Il existe un moyen de connaitre le nombre de sous-composants d'un élément:

```
nb_sous_elements = Request.Form("elem").Count
```

Pour la méthode "get" il faut utiliser **Request.QueryString** . Cette méthode permet de récupérer tous les arguments mis au bout d'une URL.

Il est aussi possible d'utiliser le rang de l'élément dans le formulaire plutôt que son nom.

Attention au premier chargement

Faites attention si au premier chargement d'une page vous voulez initialiser votre formulaire avec des valeurs par défauts. Utilisez par exemple des variables de session, ne vous fiez pas aux valeurs initiales des éléments du formulaire.

V-D - Petite disgression sur les arguments passés par les URL

Pour passer des valeurs via une URL, la syntaxe est simple et tout se met dans la partie "href" du lien:

```
href="nomdelapage.asp?var1=Valeur1&var2=Valeur2&...&varN=ValeurN"
```

où var1 à N sont les noms des variables et Valeur1 à N leurs valeurs.

Ensuite utilisez:

```
<% var1=Request.QueryString("var1")
var2=Request.QueryString("var2")
..
varN=Request.QueryString("varN") %>
```

Tout l'intérêt de passer des paires noms/valeur est que vous pouvez construire dynamiquement des liens vers une page en transmettant des paramètres, sans recourir à un formulaire (exemple: vous créer dynamiquement un tableau avec un liste de produits, avec à chaque ligne un lien pour accéder aux prix particuliers du visiteur).

V-E - Mise en oeuvre

Chaque élément à sa méthode pour être "repositionné" et pour renvoyer sa valeur.

Pour chaque type d'élément je vais vous montrer comment récupérer et repositionner sa valeur.

Reportez vous à la syntaxe détaillée de chaque élément pour avoir toutes les options possibles (size,maxwidth,etc.).

Une visite à cet endroit vous sera profitable:

Référence HTML par cyberzoïde

V-E-1 - Case à cocher

On récupère "on" ou "off" à travers le nom de la case, il faut préciser "checked" pour recocher et rien pour ne pas recocher.

Utilisation de cases à cocher

```
<html><head><title>Ma page ASP</title></head>
<% cocher=request.form("cocher")%>
<form method="post" name="formulaire" action="page.asp">
Case à cocher <input type="checkbox" name="cocher"
<%if cocher="on" then response.write " checked"%> >
<input type="submit">
</form>
</body></html>
```

L'état initial de la case à cocher est "décoché". Attention, si la case est décochée on récupère une valeur vide et pas "off". Attention aussi à ce que "checked" ne soit pas collé au nom de l'élément (d'où l'espace mis volontairement).

V-E-2 - Champ de saisie / Champ de saisie de fichier / Champ de mot de passe

On récupère directement sa valeur à travers son nom.

Utilisation de champs de saisie

```
<html><head><title>Ma page ASP</title></head>
<% champ=request.form("champ")%>
<form method="post" name="formulaire" action="page.asp">
Champ de saisie <input type="text" name="champ" value="<%=champ%>" >
<input type="submit">
</form>
</body></html>
```

En précisant une valeur avec "value", on récupère ce qui a été saisi. Evidemment cette valeur peut être fabriquée, contrôlée, modifiée,etc.

Un champ de saisie de nom de fichier est un champ de saisie dont la valeur est remplie après avoir cliqué le bouton "Parcourir".

Un champ mot de passe est un champ de saisie où les caractères tapés sont remplacés par des étoiles.

Le champ de saisie de nom de fichier ouvre le dialogue standard de sélection de fichier sur votre système.

Le formulaire peut être modifié pour déclencher l'envoi d'un email avec pièce jointe, par le logiciel de courrier du poste du client, à l'aide d'un champ de saisie de nom de fichier, reportez vous au **cours de Hugo ETIEVANT** pour un exemple détaillé.

V-E-3 - Champ caché

Un champ caché est identique à un champ de saisie, mais il n'est pas affiché (son type est "hidden" et pas "text").

Son utilité est de pouvoir stocker des valeurs propres à l'application sans qu'elles soient affichées (et donc modifiables).

V-E-4 - Liste déroulante

Utilisation de listes déroulanets

```
<html><head><title>Ma page ASP</title></head>
<% liste=request.form("liste")%>
<form method="post" name="formulaire" action="page.asp">
Liste déroulante <select name="liste" >
<option value="Un"
<% if liste="Un" then response.write "selected"%> >Un</option>
<option value="Deux"
<% if liste="Deux" then response.write "selected"%> >Deux</option>
<option value="Trois"
<% if liste="Trois" then response.write "selected"%> >Trois</option>
</option>
<input type="submit">
</form>
</body></html>
```

Il faut préciser "selected" dans la partie "option" pour que la ligne soit sélectionnée.

Evidemment vous n'écrirez jamais toutes les lignes à la main, vous utiliserez des boucles avec des tableaux ou des résultats d'interrogations de base de données pour construire la liste et resélectionner la ligne choisie.

Chaque ligne "option" rajoutée vous rajoute une ligne dans la liste. Cet élément sera beaucoup utilisé dans la partie "Utiliser une base de données".

Cas d'une liste à sélection multiple:

Utilisation de listes à sélection multiple

```
<html>
<head>
<title>Ma page ASP</title>
</head>
```

Utilisation de listes à sélection multiple

```
<form method="post" name="formulaire" action="page.asp">
Liste déroulante <select name="liste" multiple>
<option value="Un"
<% for i=1 to request.form("liste").count
if request.form("liste")(i)="Un" then response.write  "selected"
next %> >Un</option>
<option value="Deux"
<% for i=1 to request.form("liste").count
if request.form("liste")(i)="Deux" then response.write "selected"
next %> >Deux</option>
<option value="Trois"
<% for i=1 to request.form("liste").count
if request.form("liste")(i)="Trois" then response.write "selected"
next %> >Trois</option>
<option value="Quatre"
<% for i=1 to request.form("liste").count
if request.form("liste")(i)="Quatre" then response.write "selected"
next %> >Quatre</option>
</option>
<input type="submit">
</form>
</body>
</html>
```

Cet exemple ne vaut que pour comprendre la façon de récupérer les choix, vous trouverez de bien meilleures façons de coder ce genre de choses.

V-E-5 - Boutons radio

Les boutons radios sont un groupe de boutons dont un seul à la fois peut être coché, ils sont mutuellement exclusifs.

Chaque groupe est nommé et chaque bouton à une valeur, on récupère à travers le nom du groupe la valeur du bouton coché.

Plusieurs groupes de boutons sont possibles.

Utilisation de boutons radio

```
<html>
<head>
<title>Ma page ASP</title>
</head><% groupe=request.form("groupe")%>
<form method="post" name="formulaire" action="page.asp">
choix un <input type="radio" name="groupe" value="un"
<% if groupe="un" then response.write "checked"%> ><br>
choix deux <input type="radio" name="groupe" value="deux"
<% if groupe="deux" then response.write "checked"%> ><br>
choix trois <input type="radio" name="groupe" value="trois"
<% if groupe="trois" then response.write "checked"%> ><br>
<input type="submit">
</form>
</body>
</html>
```

Attention, initialement aucune case n'est cochée.

V-E-6 - Champ zone de texte

Le champ zone de texte (textarea) permet la saisie de texte multilignes.

Attention, les sauts de lignes peuvent être écrits "en dur" dans le texte au moment où la valeur sera transmise, ou les sauts de lignes peuvent se faire en fonction de la taille de la zone (option "wrap").

Utilisation de zone de texte
```
<html>
<head>
<title>Ma page ASP</title>
</head>
<% zonetexte=request.form("zonetexte")%>
<form method="post" name="formulaire" action="page.asp">
<textarea name="zonetexte">
<%=zonetexte%>
</textarea>
<input type="submit">
</form>
</body>
</html>
```

Pour remettre la valeur saisie, il suffit de l'écrire à l'intérieur de la balise.

V-E-7 - Boutons et images

Certains boutons permettent de soumettre le formulaire, d'autres d'effacer son contenu, et les images soumettent le formulaire.

Les boutons de soumission du formulaire sont nommés, on récupère leurs valeur aux travers de leurs noms.

Utilisation des boutons de valdiation
```
<html>
<head>
<title>Ma page ASP</title>
</head>
<% action=request.form("action")
if action="" then texte="Pas d'action."
if action="Lancer" then texte="Lancement choisi."
if action="Abandonner" then texte="Action abandonnée."%>
<%=texte%><br>
<form method="post" name="formulaire" action="page.asp">
<input type="submit" name="action" value="Lancer">
<input type="submit" name="action" value="Abandonner">
</form>
</body>
</html>
```

Le bouton reset ne provoque pas la soumission du formulaire, il repositionne les éléments du formulaire à leurs valeurs initiales (pour ceux qui en ont une, et si c'est applicable, ce qui n'est pas le cas de "textarea").

Utilsiation de bouton reset
```
<html>
<head>
<title>Ma page ASP</title>
</head><
form method="post" name="formulaire" action="page.asp">
<input type="texte" name="champ" value="Valeur initiale">
<input type="submit" name="action" value="Lancer">
<input type="reset" name="raz" value="Effacer">
</form>
```

Utilsiation de bouton reset
```
</body>
</html>
```

Le bouton image permet d'afficher une image au lieu du texte. On peut aussi récupérer les coordonnées du point où on a cliqué.

Utilisation de boutons images
```
<html>
<head>
<title>Ma page ASP</title>
</head>
<% x=request.form("go.x")
y=request.form("go.y")%>
x: <%=x%><br>
y: <%=y%><br>
<form method="post" name="formulaire" action="page.asp">
<input type="texte" name="champ" value="Valeur initiale">
<input type="submit" name="action" value="Lancer">
<input type="image" name="go" src="go.gif">
<input type="reset" name="raz" value="Effacer">
</form>
</body>
</html>
```

Si x et y sont vides, c'est qu'on a pas cliqué sur le bouton.

V-F - Conclusion

En utilisant "savamment" les formulaires, vous pourrez créer une interface efficace et agréable dans vos pages.

L'utilisation des styles CSS est recommandée pour en plus avoir des "belles" pages.

N'oubliez pas de consulter la référence de la syntaxe des éléments de formulaires pour en tirer pleinement partie.

VI - Utiliser une base de données

Dans les exemples, un mot en italique représente un paramètre à fournir, les crochets ([]) indiquent un paramètre optionnel.

VI-A - Créer une connexion à une base de données

Pour se connecter à une base de données il ne faut pas oublier les points suivants:

- vous devez connaitre les paramètres de la connexion (nom de la source de données, utilisateur, mots de passe,etc.)
- vous devez avoir les droits d'accès nécessaires et suffisants pour utiliser cet accès (droits d'écriture sur les fichiers de la base de données par exemple).

Les connexions se font à l'aide de l'objet "Connection", qui vous retournera un "identifiant" vous permettant d'exploiter votre base de données.

Exemple de connexion à une base de données nommée "myconn" par ODBC:

```
Set cnx = Server.CreateObject("ADODB.Connection")
cnx.Open "myconn"
```

Il est maintenant possible, à l'aide de "cnx", d'utiliser la base de données "pointée" par la source ODBC située sur le serveur et nommée "myconn".

Attention lors de l'utilisation d'une source de données ODBC à bien la paramétrer. Si (par exemple) cette source accède à la base de données avec un utilisateur qui n'a que les droits de lecture, vous ne pourrez faire aucune insertion/suppression.

La description exhaustive des connexions possibles et de leur mise en oeuvre est en dehors du champ de ce tutoriel.

Une connexion une fois ouverte est valable jusqu'à la fin de la page. Il n'est pas utile de laisser une connexion ouverte pendant toute la durée de la session de l'utilisateur, le "pool" de connexion de votre serveur se chargera d'optimiser les accès.

Une connexion se ferme simplement:

```
cnx.Close
```

et la connexion est fermée, puis si on n'a plus besoin de cet objet :

```
set cnx = nothing
```

VI-B - Exécuter des commandes

Les commandes se font dans le langage de la base de données (généralement en Sql).

Pour éxécuter une commande sur la base de données, on utilise la syntaxe:

```
Set Recordset = Connexion.Execute(Commande, [NBaffectés],[Options])
```

"Recordset" est un paramètre optionnel, il permet de récupérer le résultat de l'éxécution de la commande. Il sera vide si la commande ne retourne rien. Il est donc inutile de créer un recordset si la commande n'est pas une requête de sélection -> Connexion.Execute(Commande,[Nbaffectés],[Options]).

"Commande" est l'instruction que vous voulez faire éxécuter. Cette instruction sera quelque chose du style "delete from tableX", "insert into cible..."," select * from source",etc.

"NBaffectés" est un paramètre optionnel, qui permet à la couche de connexion à votre base de données de vous retourner le nombre de lignes affectées.

"Options" permet de préciser le type de commande exécutée (commande sql simple, procédure stockée, exécution asynchrone,etc.).

Pour gagner du temps sur l'exécution de commandes ne retournant pas de valeurs (ou si les valeurs ne vous intéressent pas), l'option adCmdExecuteNoRecords permet de supprimer le retour du jeu d'enregistrements. Ce jeu d'enregistrements est systématiquement créé, à moins que l'option adCmdExecuteNoRecords ne soit précisée.

VI-C - Récupérer des jeux d'enregistrements

Une base de données est (généralement) organisée en tables, contenant des lignes qui sont découpées en colonnes. On récupère les enregistrements sous forme de lignes (un peu comme une portion de fichier d'un tableur) dans ce qu'on appelle un "jeu d'enregistrements" (recordset).

Pour travailler sur des enregistrements, vous devez avoir une connexion ouverte, créer un objet RecordSet (jeu d'enregistrements) et le remplir à l'aide d'une commande sql. La syntaxe générale est:

```
Recordset.Open(Source,Connexion,TypeCurseur,TypeVerrou,Options)
```

"Source" est un objet "Command" valide, une commande Sql , un nom de table,une procédure stockée,etc.

"Connexion" est un objet "connexion" ouvert ou une chaine de connexion

"TypeCurseur" définit le type de curseur qui sera retourné. Suivant le type, les déplacements dans le jeu d'enregistrement seront possibles seulement séquentiellement "en avant" ou "en avant et en arrière", directement à une position définie, etc. La valeur par défaut est "séquentiellement en avant seulement".

"TypeVerrou" précise si le jeu d'enregistrements est en lecture seule ou non, et le type de verrouillage. La valeur par défaut est "verrouillé en lecture seule".

"Options" permet de préciser le type de commande (table, procédure stockée), comment récupérer les enregistrements,etc.

Exemple basique de récupération de données:

```
sql = "select * from produits"
Set RS = Server.CreateObject("ADODB.RecordSet")
RS.Open sql, cnx
```

Un jeu d'enregistrement se ferme avec sa méthode close (RS.close).

Laisser ouverts des jeux d'enregistrements devenus inutiles (créer sans jamais réutiliser ou fermer) peut encombrer le serveur, à l'extrême (cas de recordsets importants) peut bloquer l'exécution.

VI-D - Exploiter les jeux d'enregistrements

RS désigne un "RecordSet" dans toute cette partie.

L'objet Recordset est très riche, mais on se concentrera sur l'essentiel pour l'exploiter rapidement.

VI-D-1 - Comment savoir le nombre de colonnes retournées

C'est la propriétés RS.Fields.Count , sachant que la première colonne a le numéro 0 , la dernière est indexé (porte le numéro) à RS.Fields.Count-1.

Cette propriété peut être inaccessible suivant le type de curseur (celui par défaut notamment).

VI-D-2 - Comment accéder à une colonne de l'enregistrement courant

Deux options:

```
valeur = RS(i)
```

où "i" est le numéro de la colonne (en commençant à 0)

```
valeur = RS("nomcol")
```

où "nomcol" est le nom d'une colonne retournée.

La méthode RS("nomcol") permet de savoir précisément ce que l'on fait, donne un code lisible et qui fonctionnera même si l'ordre des colonnes change. La méthode RS(i) est plus risquée, notamment si l'ordre des colonnes change, mais en terme de performance est nettement meilleure.

VI-D-3 - Comment avoir le nom de la n-ième colonne de l'enregistrement courant

Très utile quand on ne sait pas l'ordre des colonnes à l'avance:

```
nomcol=RS(i).name
```

La première colonne est la numéro 0.

VI-D-4 - Se déplacer dans un jeu d'enregistrement

Certaines opérations (ex: moveprevious) requièrent un curseur d'un type approprié.

Avancer d'un enregistrement : RS.movenext

Reculer d'un enregistrement. RS.moveprevious

Aller au début du jeu d'enregistrement : RS.movefirst

Aller à la fin du jeu d'enregistrement : RS.movelast

VI-D-5 - Savoir où l'on se trouve

Si on est à la fin : RS.eof est **True** (valeur booléenne vraie).

Si le jeu d'enregistrement est vide : RS.eof et RS.bof sont **True** (valeur booléenne vraie) .

VI-E - Petit exemple

Comment construire une listbox à partir d'une table où chaque ligne contient un code et un nom:

```
<HTML>
<HEAD>
<TITLE>Exemple simple</TITLE>
<BODY>
<% 'Ouvrir la source de données
Set cnx = Server.CreateObject("ADODB.Connection")
cnx.Open "myconn" %>
<form name="formulaire" method="post" action="page.asp">
<select name="choix">
<% sql="select code,nom  from table"
Set RS = Server.CreateObject("ADODB.RecordSet")
RS.Open sql, cnx,adOpenKeyset
do while not RS.eof %>
<option value="<%=RS("code")%>" ><%=RS("nom") %></option>
<% RS.movenext
loop
RS.close
cnx.close
set cnx = nothing
set RS = nothing%></select>
</form>
</BODY>
</HTML>
```

Le fichier adovbs.inc contient toutes les constantes à utiliser pour les types et les options. Il est souvent situé dans Program Files\Fichiers Communs\System\ADO.

VII - Utiliser les cookies

VI-A - Qu'est-ce que les cookies

Les cookies sont des informations stockées de manière persistante sur l'ordinateur du visiteur d'un site web. Il se présentent (généralement) sous la forme d'un ou plusieurs fichiers texte.

Leur fonction est de permettre le stockage, d'une visite sur l'autre, d'informations telles que : identité de la personne, mot de passe, date de la dernière visite, etc.

L'application la plus courante est le stockage de l'identification de connexion des visiteurs, afin de leur éviter la resaisie systématique de leur mot de passe.

Un cookie peut être de type simple (un nom, une valeur), ou être de type dictionnaire (un ensemble de paire clé/valeur).

VI-B - Limitations

C'est le navigateur employé, et après lui l'ordinateur qui le fait fonctionner, qui vont décider de la manière dont vous pourrez utiliser les cookies.

Certains navigateurs stockent les cookies dans un fichier "cookies.txt". Si vous changez les attributs de ce fichier pour le mettre en lecture seule, aucun cookie ne pourra être écrit.

La plupart des navigateurs permettent de restreindre l'usage des cookies par les sites visités. Il n'est donc pas toujours certain que vous puissiez écrire le cookie souhaité sur le poste client, surtout si ces options sortent du cadre "habituel", typiquement si le cookie que vous voulez créer est lisible par d'autres sites que le votre.

VI-C - Fonctions disponibles

Les cookies, en ASP, sont en écriture une "collection" de l'objet *Response*, et en lecture de l'objet *Request*.

En ASP 2.0, cette collection possède 5 propriétés:

Domain : Indique si le cookie est seulement retourné aux pages à l'intérieur du domaine où il a été créé. La valeur par défaut est le domaine courant de la page. Vous ne devriez changez cette valeur que si vous voulez spécifier l'étendue de la "visibilité" du cookie. *Propriété en écriture seulement*

Expires : Fixe la date d'expiration du cookie. Si cet attribut n'est pas fixé à une date/heure ultérieure à la date/heure courante, le cookie expirera quand le navigateur sera fermé. *Propriété en écriture seulement*

HasKeys : Précise si le cookie est un objet dictionnaire (Dictionary). *Propriété en lecture seulement*

Path : Si cette valeur est fixée (nom de chemin de fichier), le cookie est uniquement renvoyé en réponse aux accès aux pages à l'intérieur de ce chemin. *Propriété en écriture seulement*

Secure : Précise si le cookie est "sécurisé". Un cookie sécurisé est envoyé uniquement à travers le protocole HTTPS. Les cookies "non sécurisés" peuvent être envoyés via HTTP ou HTTPS indifféremment. Dans les deux cas, aucun cryptage n'est fourni, le cookie est stocké en clair. *Propriété en écriture seulement*

VI-D - Les utiliser

La syntaxe générale est:

Affectation : Request.Cookies("nom_cookie")["clé"] = valeur

Lecture : valeur = Response.Cookies("nom_cookie")["clé"]

VI-E - Exemples pratiques

Créer le cookie "nom" qui expire 30 jours après sa création, en lui affectant la valeur de la variable nom.

```
Response.Cookie("nom")=nom
Response.Cookie("nom").Expires = Date + 30
```

Créer un cookie de type dictionnaire, avec pour chaque "visiteur" deux informations

```
Response.Cookie("visiteur")("nom")=nom
Response.Cookie("visiteur")("ville")=ville
Response.Cookie("visiteur").Expires = Date + 30
```

*N'oubliez pas **qu'aucun** cryptage n'est fourni, à vous de trouver un moyen de crypter vos données si vous ne voulez pas qu'elles soient écrites et qu'elles transitent en clair.*

VIII - Utiliser les fichiers

VIII-A - Comment Asp gère l'accès aux fichiers

L'accès aux fichiers se fait à l'aide de l'objet FileSystemObject.

Il s'agit d'un objet de script, disponible via la bibliothèque d'exécution de script (fichier scrrun.dll) qui est installée par défaut avec les moteurs de script Active Scripting.

L'objet FileSystemObject donne accès au système de fichiers côté serveur ou côté client.

Il permet à travers ses méthodes, propriétés et collections de parcourir les dossiers et fichiers de l'ordinateur, de lire ou d'écrire dans les fichiers, de créer, déplacer, supprimer des dossiers ou des fichiers, d'obtenir des informations sur les disques, dossiers, fichiers...

VIII-B - Collections et méthodes de l'objet FileSystemObject

Elles sont très nombreuses et ne peuvent tout être abordées.

Pour une référence plus complète reportez vous à la **documentation Msdn**.

VIII-C - Accès aux fichiers : attention au chemin, aux permissions

Pour accéder à un fichier, il faut fournir son chemin absolu sur le serveur.

La fonction Server.MapPath() permet d'obtenir le chemin physique d'un fichier.

Exemple: Server.MapPath("/monsite/mapage.asp") pourrait vous rendre "e:\www\monsite\mapage.asp".


Pour pouvoir modifier/supprimer des fichiers, des répertoires, les processus de IIS doivent avoir les droits suffisants, et l'option "Ecriture" doit être coché pour le site.

VIII-D - Exemples d'utilisation

VIII-D-1 - Créer un fichier

L'exemple suivant crée un fichier contenant deux lignes.

Création simple de fichier
```
<%
set fs=CreateObject("Scripting.FileSystemObject")
set fichier=fs.CreateTextFile(Server.MapPath("/monsite/essai.txt"),true,false)

fichier.writeline "Une ligne"
fichier.write "Deux ligne"
fichier.close
%>
```

Voici les paramètre de CreateTextfile :

- nom du fichier : nom de fichier à créer
- écrasement : true on écrase un fichier existant, false on n'écrase pas
- unicode : true on crée un fichier Unicode, false un fichier Ascii

Pour écrire dans le fichier, la méthode write écrit une ligne sans rajouter de sans de ligne, alors que writeline en insère un.

VIII-D-2 - Lire un fichier

Pour lire un fichier, il faut obtenir un descripteur sur ce fichier et l'ouvrir comme un objet textstream.

Lire un fichier

```
<%
 set object=fs.GetFile(Server.MapPath("/monsite/" & "\\monfichier.txt")
 set desc=object.OpenAsTextStream(1,-2)
%>
```

Les paramètres de OpenAsTextStream sont:

premier paramètre, mode de lecture :

- 1 ou ForReading : ouverture en lecture seule
- 2 ou ForWriting : ouverture en écriture (écrasement)
- 8 ou ForAppending : écriture à la fin du fichier

deuxième paramètre, format du fichier:

- 0 ouTristateFalse : ouvre le fichier en Ascii (valeur par défaut)
- -1 ou TristateTrue : ouvre le fichier en Unicode
- -2 ou TristateUseDefault : ouvre le fichier avec la valeur système par défaut

VIII-D-3 - Parcourir tous les fichiers d'un dossier

La méthode "for each" est très précieuse dans les parcours de collection, et notamment pour gérer les dossiers/fichiers.

Parcours des fichiers d'un dossier

```
<% set fs=CreateObject("Scripting.FileSystemObject")
set dossier=fs.GetFolder(Server.MapPath("/monsite/source/"))
for each fichier in dossier.Files
 set obj=fs.GetFile(Server.MapPath("/monsite/source/") & "\\" & fichier.Name)
 'Traitement
next
%>
```

VIII-D-4 - Conclusion sur l'utilisation du système de fichiers

Vous trouverez dans "FileSystemObject" des collections permettant d'accéder aux disques, dossiers, fichiers, et des méthodes permettant de copier, créer, supprimer, renommer, tester l'existance d'objets.

L'objet "FileSystemObject" vous permettra de travailler sur vos fichiers et dossiers de manière pratique, pour peu que vous soyez précautionneux dans son emploi et que vous ayez les droits d'accès suffisants.

Annexe A - Optimiser ses pages ASP

Annexe A-A - Evitez les transitions HTML/ASP

Chaque transition entre un bloc "<% %>" et du code HTML consomme du temps lors de la réalisation de la page.

L'exemple le plus courant est la réalisation d'un tableau à l'aide d'une requête: si à chaque tour de boucle vous repassez en HTML juste pour écrire "<TR><TD>", autant utiliser un Response.Write.

Annexe A-B - Ne gardez pas d'objets en mémoire inutilement

Fermez et détruisez les Recordset dès que vous n'en avez plus besoin, et essayez de ne pas les garder ouvert inutilements. Un excès de recordsets pourrait provoquer l'échec de l'éxécution de votre page.

Une fois créé, un Recordset peut être fermé puis réouvert sans avoir besoin d'être détruit (avec set ...=nothing), alors réutilisez vos recordset au maximum.

Fermez et détruisez les connexions quand vous n'en avez plus besoin (avec set ... = nothing).

Annexe A-C - Utilisez efficacement les jeux d'enregistrements

Evitez les "select *", ne récupérez que ce qui vous est nécessaire, un jeu d'enregistrement plus petit consomme moins de ressources.

Pour accéder aux valeurs, préférez la syntaxe RS(0) plutôt que RS("nom"). L'accès direct par numéro de rang est beaucoup plus rapide.

Quand vous éxécutez une commande qui ne devrait pas retourner de jeu d'enregistrements, pensezà utiliser la constante "adExecuteNoRecords".

Annexe A-D - Attention aux variables de sessions et d'application

Elles occupent une place non négligeable en mémoire. Avec un grand nombre de sessions, on risque des problèmes de fonctionnement du serveur.

Elles ne sont pas supposées contenir de grosse quantité de données. Utilisez une base de données à la place.

Annexe A-E - Choisissez les bons Request

Utilisez "Request.Form" pour récupérer les valeurs d'un formulaire, "Request.Querystring" pour les valeurs d'une URL, et évitez la forme "Request" seule. Cette syntaxe oblige le serveur à parcourir toutes les collections pour chercher votre valeur, ce qui dégrade les performances.

Si vos paramètres peuvent venir d'un formulaire ou d'une URL, essayez d'abord une méthode et faites un test pour savoir si vous devez essayer la deuxième.

Annexe A-F - Utilisez Response.Buffer

En déclarant "Response.buffer = true" (ou en activant le buffer pour toutes vos pages), le contenu de la page n'est envoyé au navigateur qu'en fin d'éxécution, ou quand vous utilisez "Response.Flush".

L'utilisation du buffer peut vous faire gagner jusqu'à 10% de performance. N'abusez cependant pas du "Flush", car les performances se dégraderaient vite.

Annexe A-G - Utilisez "Option explicit"

En précisant "Option Explicit" (qui doit être la première instruction de la page ASP), vous êtes obligés de déclarer toutes les variables, mais vous pouvez obtenir un gain de performances jusqu'a 10% sur votre page.

Toutes les variables devront être déclarées avec "dim".

Annexe A-H - Utilisez les procédures stockées dans votre base de données

Si vous pouvez, utilisez les procédures stockées, ce qui est substantiellement plus rapide que de resoumettre une requête que votre base de données devra réanalyser.

Annexe A-I - Note finale

Tous ces conseils ne produiront peut-être que peu d'effet sur une "petite" application, mais avec une forte charge et beaucoup d'utilisateurs, un code optimisé sera nettement plus efficace. Sans oublier qu'un mauvais code peut donner une page qui ne s'éxécutera jamais, par manque de ressources du serveur.