Dotnet France
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

Les bases fondamentales de Microsoft ASP .NET Ajax

Version 1.0

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

Sommaire

1 Introduction

1.1 Présentation

Ce chapitre s'adresse à des personnes, maîtrisant la conception et le développement d'applications Web, avec la technologie ASP .NET. Aussi, il met l'accent sur le Framework Ajax côté serveur. Le côté client sera abordé dans un autre chapitre.

1.2 Rappels sur la technologie ASP .NET

La technologie ASP .NET est une technologie proposée par Microsoft, permettant de développer des applications Web. En 2005, Microsoft a proposé la technologie ASP .NET 2.0, qui a apporté d'importantes avancées par rapport aux versions précédentes :

- Dans la conception des projets Web : création d'une application Web ou d'un site Web
- Dans le développement : les pages maîtres, gestion de l'état des données, propositions de services d'application (gestion de la sécurité, de la navigation, des dépendances de cache vers les bases de données, de gestion des profils utilisateurs…).
- De nouveaux contrôles ASP .NET.
- Dans le déploiement via Visual Studio : copie et synchronisation via l'outil de copie de sites Web, et la publication avec la possibilité de compiler partiellement ou quasi-complètement les sources de l'application…

Cependant, d'importants flux de données naviguent entre les serveurs et les clients Web, et les performances côté client restent limitées (nativement, pas d'envoi de requête HTTP en mode asynchrone…), et la conception de contrôles utilisateurs sophistiquées est aisée… C'est pourquoi, Microsoft propose une implémentation des concepts d'Ajax (**A**synchronous **J**avaScript **A**nd **X**ML) :

- Sous forme d'extensions depuis la version 2.0 du Framework .NET.
- Intégrée dans le Framework .NET depuis la version 3.5.

1.3 Utilité

Microsoft ASP .NET Ajax peut être utilisé à des fins diverses :

- Validation et mise à jour d'une partie d'une page
- Aide à la saisie (listes …)
- Lecture de flux RSS
- Tri, filtrage et réorganisation de données côté client
- Applications de gestion de documents
- Chargement progressif de données volumineuses
- Consommation de services distants
- …

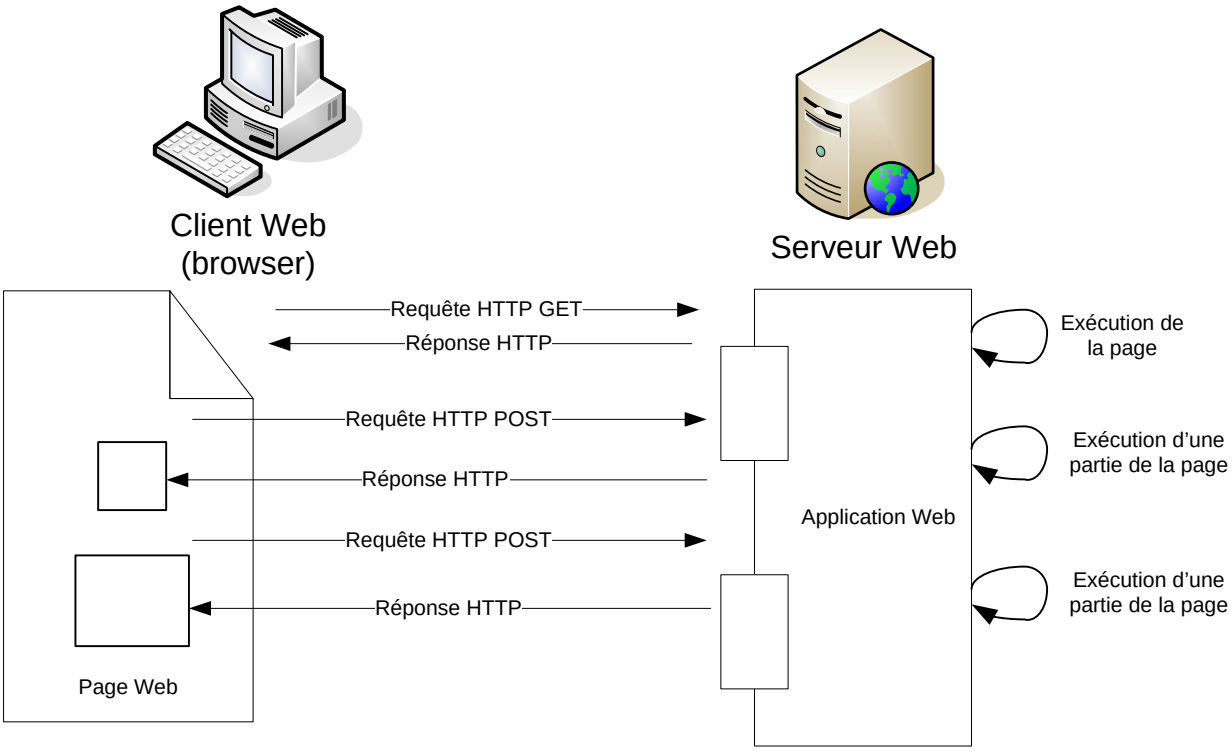
2 Les bases fondamentales d'ASP .NET Ajax

2.1 Echanges client/serveur avec ASP .NET (« classiques »)

Dans une application ASP .NET classique, un client Web envoie des requêtes HTTP au serveur Web. Chacune des requêtes est exécutée de manière synchrone. Voici un exemple de scénario :

- Le client envoie une première requête au serveur, et patiente.
- Le serveur traite cette requête, exécute la page ASP .NET visée, et renvoie le résultat de l'exécution au client.
- Le client clique par exemple sur un bouton, une nouvelle requête est alors envoyée au serveur Web. Il patiente de nouveau, pendant que le serveur traite sa requête.
- Une fois la réponse obtenue, le client voit dans son navigateur le résultat de l'exécution, et peut de nouveau envoyé une nouvelle requête.

Voici un schéma explicitant ce scénario :



2.2 Echanges client/serveur avec ASP .NET Ajax

Avec ASP .NET Ajax, le client pourra envoyer des requêtes HTTP, qui mettront à jour une ou plusieurs parties d'une même page. Ces requêtes pourront s'exécuter de manière asynchrone (« en même temps »).

2.3 Format d'échange de données

Actuellement (en 2009), le langage d'échanges de données standard inter-applicatif est le langage XML. Cependant, dans ASP .NET Ajax, ce langage pose des problèmes en matière de performance :

- Le langage XML est un langage verbeux (balises d'ouverture et de fermeture, répétition d'éléments, …). Les données échangées entre le client et le serveur sont donc volumineuses.
- Un flux XML n'est pas interprété nativement par le langage JavaScript. Ainsi, pour accéder aux données du flux, il est nécessaire de parser ce flux et créer un arbre DOM.

Fort de ces constats, dans ASP .NET Ajax, les échanges de données (par défaut, car il est possible de modifier ce comportement) se font avec le langage JSON (**J**ava**S**cript **O**bject **N**otation). Ce langage permet d'écrire des flux moins volumineux, et est nativement interprété par le langage JavaScript. Pour vous montrer la différence entre ces deux langages, voici deux flux de données :

```
' Flux de données au format XML

<?xml version="1.0" ?>
<menubar>
  <menu name="File">
    <command value="New" action="CreateDoc" />
    <command value="Open" action="OpenDoc" />
    <command value="Close" action="CloseDoc" />
  </menu>
</menubar>
```

```
' Flux de données au format JSON

[ {
"menu": "File",
"commands": [
 {
    "value":"New",
    "action":"CreateDoc"
 },
 {
    "value":"Open",
    "action":"OpenDoc"
},
{
    "value":"Close",
    "action": "CloseDoc"
 } ]
 } ]
```

Si vous souhaitez en savoir plus sur ce langage, vous pouvez consulter le site officiel de JSON, qui publie ses spécifications : www.json.org

3 Présentation d'ASP .NET Ajax

3.1 Vue d'ensemble

Ajax n'est pas une technologie propre à Microsoft, qui en propose toutefois une implémentation, afin d'étendre sa technologie permettant de développer des applications Web, la technologie ASP .NET :

- Microsoft ASP .NET AJAX Library : framework JavaScript (nouveaux éléments de syntaxe, bibliothèque de classes complémentaire)
- Microsoft ASP .NET AJAX Extensions, présentés sous la forme de nouveaux contrôles serveur ASP .NET, et classes associées
- Microsoft ASP .NET AJAX Control Toolkit : un ensemble de contrôles Web enrichis (widgets). Ces contrôles ne sont pas abordés dans ce cours. Si vous souhaitez en savoir plus, vous pouvez lire le chapitre « Ajax Control Toolkit »

La philosophie de Microsoft ASP .NET Ajax, telle qu'elle est présentée par Microsoft, est très orientée serveur : dans la continuité de la technologie ASP.NET, elle propose une implémentation basée sur des composants serveurs. Toutefois, cette implémentation est extensible, en nous laissant la possibilité de créer nos propres composants.

3.2 Microsoft ASP .NET Ajax : côté serveur

Voici un schéma présentant les fonctionnalités prises en charge par le Framework .NET, côté serveur :

Ces composants permettent :

- De configurer le code JavaScript généré dans la page :
  - Localisation / globalisation, de manière à internationaliser / personnaliser les textes et formats de données
  - Débogage et traçage
- De créer des services Web Ajax, des services WCF et des méthodes de page consommables dans le bloc JavaScript des pages ASP .NET
- De configurer et utiliser les services d'application ASP .NET (gestion de l'authentification des utilisateurs, des autorisations sur les ressources de l'application et gestion du profile des utilisateurs)
- De créer des contrôles serveur personnalisés, qui définissent un comportement côté client (extenders)
- …

3.3 Microsoft ASP .NET Ajax : côté client

Voici un schéma présentant les fonctionnalités prises en charge par le Framework .NET, côté serveur :

Ce framework côté client est composé de fichiers JavaScript, assurant :

- La possibilité de créer des composants côté client, avec un fort aspect orienté objet
- La prise en charge de la compatibilité dans les navigateurs Web
- Les composants réseau permettent de simplifier la communication avec l'application côté serveur
- …

4 Microsoft Ajax Library : les extensions JavaScript

Ce sujet est trop vaste pour pouvoir le traiter dans ce support de cours. Nous nous attacherons ainsi à vous présenter en quoi consiste les extensions JavaScript.

4.1 Présentation

Microsoft Ajax Library contient des fichiers d'extension .js, téléchargés sur le navigateur client à la demande, autrement dit suivant les fonctionnalités utilisés dans la page affichée. Tous ces fichiers de base sont contenus en tant que ressources incorporée dans l'assembly *System.Web.Extensions.dll* du Framework .NET.

Les extensions JavaScript enrichissent le langage JavaScript de base, tout en proposant des :

- Objets, instructions et concepts de plus haut niveau, tels que des notions objet (classes, héritage, interfaces, énumérations …).
- Reconnaissance de type d'un objet.
- Des capacités de mise en œuvre de la réflexion.

Autrement dit, Microsoft Ajax Library contient une importante couche d'abstraction sur le langage JavaScript, afin :

- De proposer des types de haut niveau, enrichissant les types natifs du langage JavaScript.
- D'implémenter dans ce langage, des concepts orientés objet.
- De proposer de nouvelles classes et objets offrant d'avantages de fonctionnalités.

Il est important de noter que Microsoft Ajax Library, fonctionne dans la plupart des navigateurs, couramment utilisés par les internautes (Internet Explorer, FireFox, Google Chrome, Opera, …)

4.2 Les types de haut niveau

Les types de haut niveau fournis par le Framework JavaScript sont les suivants :

- *Array* : étant le type Array natif, afin de pouvoir facilement énumérer et rechercher les éléments dans un tableau
- *Boolean* : étant le type Boolean natif, afin de pouvoir déduire une valeur booléenne à partir d'une chaîne de caractères
- Date : étend le type Date natif, notamment avec la prise en compte du formatage de données
- *Error* : permet de simuler une exception
- *Function* : étend le type Function natif, afin de proposer des méthodes permettant de définir des classes, espaces de noms…
- *Number* : étend le type Number natif

- Object : étend le type Object natif
- *RegExp* : encapsule le type RegExp natif
- *String* : étend le type String natif, en fournissant différentes options de formatage, de découpage et d'analyse de chaînes de caractères

4.3 L'alias *$get*

Equivalente à l'instruction JavaScript *document.getElementById()*, l'alias *$get* permet d'accéder à un contrôle de la page, via son modèle DOM. Cet alias est utilisable dans tous les navigateurs :

```
$get('idControl')
```

4.4 Les extensions orientées objet

Les extensions orientées objet apportées par le Framework JavaScript sont les suivantes :

- Création de classes
- Spécification d'espace de noms
- Mise en œuvre de l'héritage
- Utilisation des interfaces
- Utilisation d'énumérations

4.5 Exemples d'utilisation de ces extensions

4.5.1 Exemple de création d'une classe

Voici quelques exemples de mise en œuvre, des extensions proposées par Microsoft Ajax Library :

```
' Exemple de création de classes avec Microsoft Ajax Library

// Enregistrement de l'espace de noms de la classe.
Type.registerNamespace("ExtensionsJavaScript");

// Définition de la classe.
ExtensionsJavaScript.Personne = function
ExtensionsJavaScript$Personne(nom, prenom) {
    // Valorisation des attributs.
    this._nom = nom;
    this._prenom = prenom;
}

// Prototype de la classe.
ExtensionsJavaScript.Personne.prototype = {
    toString : function toString () {
        return this._nom + ' ' + this._prenom;
    },

    get_nom : function () {
        return this._nom;
    },

    get_prenom : function () {
        return this._prenom;
    }
}

// Enregistrement de la classe.
ExtensionsJavaScript.Personne.registerClass("ExtensionsJavaScript.Personn
e");
```

```
' Exemple de création d'une énumération

Type.registerNamespace("ExtensionsJavaScript");

ExtensionsJavaScript.Couleurs = function(){};
ExtensionsJavaScript.Couleurs.prototype =
{
    Rouge: 0xFF0000,
    Bleu:  0x0000FF,
    Vert:  0x00FF00,
    Blanc: 0xFFFFFF
}
Type.registerEnum("ExtensionsJavaScript.Couleurs");
```

```
' Exemple de mise en œuvre de l'héritage

// Définition de la classe Employe.
ExtensionsJavaScript.Employe = function ExtensionsJavaScript$Employe(nom,
prenom, service)
{
    ExtensionsJavaScript.Employe.initializeBase(this, [nom, prenom]);
    this._service = service;
}

// Enregistrement de la classe, avec l'héritage.
ExtensionsJavaScript.Employe.registerClass("ExtensionsJavaScript.Employe"
, ExtensionsJavaScript.Personne);
```

5 Microsoft Ajax Library : les classes du noyau

Ce sujet est trop vaste pour pouvoir le traiter dans ce support de cours. Il devra être traité dans un autre support de cours. Mais on peut tout de même se poser la question suivante : comment le client fait-il pour downloader les fichiers JavaScript constituant ce Framework ?

Pour trouver une réponse à cette question, il suffit de se positionner dans le fichier de configuration de l'application ASP .NET. On trouve alors la définissant un handler, qui peut être utilisé à distance, par des blocs de code JavaScript. Voici la définition de ce handler :

```
<httpHandlers>
  <remove verb="*" path="*.asmx"/>
  <add verb="GET,HEAD" path="ScriptResource.axd"
       type="System.Web.Handlers.ScriptResourceHandler,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35"
       validate="false"/>
</httpHandlers>
```

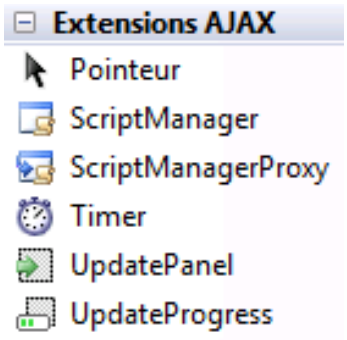
Par ailleurs, l'utilisation de ce handler est visible dans le code source XHTML, que le serveur retourne au client. En voici un exemple :

```
<script src="/DF-ExtensionsAjaxCoteServeur-
VB/ScriptResource.axd?d=4_dOjNXd0PPvxMzoLqP5Ut39d3BB7vr_NoCLtaBpR7njCjZai
ou11hTzd33AXFZCDbjosECAkS8p0NaCp3_vZG78IGQOB_Yp2_gOn3lJCl81&amp;t=fffffff
f8e6551fd" type="text/javascript"></script>
```

6 Les extensions du Framework ASP .NET Ajax

6.1 Les extensions du Framework ASP .NET Ajax

Le Framework .NET propose, côté serveur, 5 contrôles ASP .NET Ajax. Ces contrôles sont disponibles dans l'onglet *Extensions Ajax* de la boite à outils de Visual Studio :



- *ScriptManager / ScriptManagerProxy* : ces deux contrôles permettent de mettre en œuvre et paramétrer, les mécanismes Ajax dans les pages ASP .NET
- *Timer* : ce contrôle fonctionne de manière analogue au contrôle *Timer* du Framework .NET, pour les applications Windows .NET. Il permet à un client Web d'exécuter des requêtes HTTP asynchrones, de manière répétitive
- *UpdatePanel* : ce contrôle est un conteneur de contrôles ASP .NET et XHTML, permettant de mettre à jour des parties d'une page ASP .NET
- *UpdateProgress* : ce contrôle permet d'afficher un message d'attente, pendant la mise à jour partielle d'une page ASP .NET

6.2 Les contrôles ScriptManager et ScriptManagerProxy

6.2.1 Le contrôle ScriptManager

Le contrôle *ScriptManager* est obligatoire dans toutes les pages, dans lesquelles il est nécessaire d'utiliser un contrôle ASP .NET Ajax, un contrôle Ajax Control Toolkit, ou tout autre mécanisme Ajax. Il permet :

- De mettre à disposition les classes et objets JavaScript utilisés dans les scripts.
- De mettre en œuvre des appels synchrones et asynchrones de services distants.
- De gérer la localisation et la globalisation des pages.
- La mise à jour partielle des pages.
- D'utiliser le service d'application de sécurité et de gestion de profils.
- D'enregistrer des nouveaux scripts JavaScript basée sur Microsoft Ajax Library.
- …

Aussi, il ne peut en exister qu'une seule instance dans une page ASP .NET. Le langage XHTML étant interprété, il doit être positionné avant tout autre contrôle AJAX dans la page :

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

Voici quelques propriétés du contrôle *ScriptManager* :

Propriétés	Description

allowCustomErrorsRedirect	Indique si les redirections d'erreurs personnalisées sont autorisées lors d'un postback asynchrone (*true* est la valeur par défaut)
asyncPostBackErrorMessage	Lit et écrit le message d'erreur à envoyer au client lorsqu'une exception non gérée se produit, lors d'un postback asynchrone
asyncPostBackSourceElementID	Contient l'identifiant du contrôle serveur qui a déclenché le postback asynchrone
asyncPostBackTimeOut	Durée en secondes, permettant de définir le timeout d'un postback exécuté de manière asynchrone
authenticationService	Permet de définir l'état du service d'authentification côté client
enablePageMethod	Indique si les méthodes de pages ASP .NET peuvent être appelées par le script côté client (false par défaut)
enablePartialRendering	Indique si le rendu partiel est autorisé pour la page (*true* par défaut)
enableScriptGlobalization	Indique si le contrôle *ScriptManager* prend en charge via du code JavaScript, la mise en forme de données suivant une culture (false par défaut)
enableScriptLocalization	Indique si le contrôle *ScriptManager* extrait des fichiers de scripts pour la culture courante (false par défaut)
isDebuggingEnabled	Indique si le code JavaScript généré contient des informations et instructions de débogage
isInAsyncPostBack	Indique si la requête exécutée est un postback asynchrone
profileService	Permet de définir l'état du service de profils d'utilisateurs côté client
scriptMode	Permet de définir les scripts à charger, suivant le mode : Auto (par défaut), *Inherit*, *Debug* ou Release
supportsPartialRendering	Indique si un navigateur ou une version de navigateur peut supporter le rendu partiel d'une page. Si cette propriété vaut false, aucun rendu partiel ne sera effectué, indépendamment de la valeur de propriété *EnablePartialRendering*
getCurrent	Méthode statique permettant d'obtenir l'instance du contrôle *ScriptManager* de la page en cours

registerPostBackControl / RegisterAsyncPostBackControl	Permet au contrôle spécifié de déclencher un postback synchrone / asynchrone depuis un panneau modifiable
registerClientScriptBlock	Méthode statique permettant d'ajouter des blocs d'instructions JavaScript dans le rendu d'une page
registerExtenderControl	Permet d'enregistrer un extender Ajax dans la page
registerHiddenField	Méthode caché permettant d'ajouter un champ caché dans la page (contrôle input de type hidden)
registerOnSubmitStatement	Méthode statique qui assure que le script côté client associé à l'évènement OnSubmit du formulaire est émis dans un rendu partiel de la page
setFocus	Permet de spécifier le positionnement du focus vers le contrôle spécifié, après un postback asynchrone
asyncPostBackError	Est levé quand une exception n'est pas gérée côté serveur lors de l'exécution d'un postback asynchrone
resolveScriptReference	Est levé lorsque le contrôle *ScriptManager* va résoudre une référence à un script. Il est ainsi possible de modifier l'emplacement du script

6.2.2 Le contrôle ScriptManagerProxy

Cependant, comment utiliser le contrôle ScriptManager ? Dans une application ASP .NET, il n'est pas nécessaire de l'ajouter dans toutes les pages :

- Si les pages sont des pages Web classiques, on peut encapsuler ce contrôle dans un contrôle Web personnalisé, afin de centraliser l'initialisation de ses attributs et son comportement.
- Si les pages sont des pages de contenu, il est possible d'ajouter uniquement ce contrôle dans la page maître. Toutes les pages de contenues bénéficieront automatiquement de contrôle, lors de leur exécution.

Quelque soit le scénario que vous prenez, il se peut que vous avez besoin d'accéder au contrôle *ScriptManager* dans une page ASP .NET. Le contrôle *ScriptManagerProxy* vous permettra d'y accéder :

```
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
</asp:ScriptManagerProxy>
```

Mais attention, toutes les propriétés, méthodes et évènements du contrôle *ScriptManager*, ne sont pas proposés au travers du contrôle *ScriptManagerProxy*. C'est pourquoi, on pourra préférer l'utilisation d'accesseur en lecture seule pour l'exposer.

6.3 Le contrôle UpdatePanel

6.3.1 Présentation

Un contrôle *UpdatePanel* est un conteneur de contrôles, désignant une partie d'une page, qui peut être rafraîchie de manière indépendante de la page. En effet, le contenu de ce contrôle peut être rafraîchi de manière asynchrone, lorsque l'un des contrôles contenu (ou bien même de la page) effectue un postback. Ainsi, vu qu'il permet alors d'éviter les rafraîchissements complets des pages, on constate les effets suivants :

- Réduction du scintillement des pages
- Meilleure interactivité entre l'utilisateur et l'application

Afin de pouvoir être utilisé dans les pages Web, la propriété *EnablePartialRendering* du contrôle *ScriptManager* doit être valorisée à *True* (valeur par défaut). Une page peut contenir un ou plusieurs contrôles *UpdatePanel*.

6.3.2 Propriétés

Voici quelques propriétés du contrôle UpdatePanel :

Propriété	Description
childrenAsTriggers	Indique si les postbacks des contrôles enfants provoqueront une mise à jour du contrôle *UpdatePanel*. *True* est la valeur par défaut. Valorisée à *False*, cette propriété permet d'autoriser uniquement une mise à jour dynamique
contentTemplate	Contrôle définissant le rendu du contrôle *UpdatePanel* lors de son exécution
contentTemplateContainer	Contrôle utilisé pour ajouter dynamiquement des contrôles enfants
isInPartialRendering	Indique si la mise à jour du contrôle *UpdatePanel*, fait partie de la mise à jour partielle de la page
renderMode	Indique si le rendu du contrôle est effectué en mode « block » ou « inline ». « block » est la valeur par défaut
triggers	Collection de déclencheurs, où chacun est rattaché à un évènement permettant la mise à jour du contrôle

updateMode	Permet de définir le mode de rendu du tableau, en déterminant sous quelles conditions il peut être mis à jour. Deux valeurs possibles : *always* (par défaut) et *conditional*

6.3.3 Mise en œuvre

6.3.3.1 Un premier exemple

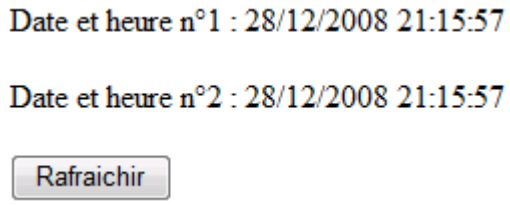
Dans une page ASP .NET :

- Ajouter un contrôle *ScriptManager*.
- Ajouter deux contrôles *Label* affichant la date et l'heure courante.
- Ajouter un boutons permettant de valorisant les deux contrôles *Label* précédemment ajoutés, avec la date et l'heure courante.

```
// C#

<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

    Date et heure n°1 : <asp:Label ID="LblDateHeure1" runat="server"
Text=""></asp:Label>
    <br />
    <br />
    Date et heure n°2 : <asp:Label ID="LblDateHeure2" runat="server"
Text=""></asp:Label>
    <br />
    <br />
    <asp:Button ID="CmdRafraichir" runat="server" Text="Rafraichir"
        onclick="CmdRafraichir_Click" />
</form>


protected void CmdRafraichir_Click(object sender, EventArgs e)
{
    string sDateHeureCourante;

    sDateHeureCourante = DateTime.Now.ToString();

    LblDateHeure1.Text = sDateHeureCourante;
    LblDateHeure2.Text = sDateHeureCourante;
}
```

```
' VB .NET

<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

    Date et heure n°1 : <asp:Label ID="LblDateHeure1" runat="server"
Text=""></asp:Label>
    <br />
    <br />
    Date et heure n°2 : <asp:Label ID="LblDateHeure2" runat="server"
Text=""></asp:Label>
    <br />
    <br />
    <asp:Button ID="CmdRafraichir" runat="server" Text="Rafraichir" />
</form>

Protected Sub CmdRafraichir_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles CmdRafraichir.Click
    Dim sDateHeure As String

    sDateHeure = DateTime.Now.ToString()

    LblDateHeure1.Text = sDateHeure
    LblDateHeure2.Text = sDateHeure
End Sub
```

On obtient le résultat suivant :



Et lors de chaque click sur le bouton Rafraichir, les deux dates sont mises à jour. Apportez au code précédent la modification suivante : mettre le second affichage de l'heure et le bouton de rafraichissement, dans un contrôle *UpdatePanel*. On obtient alors le code suivant :

```
// C#

<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

    Date et heure n°1 : <asp:Label ID="LblDateHeure1" runat="server"
Text=""></asp:Label>
    <br />
    <br />

    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            Date et heure n°2 : <asp:Label ID="LblDateHeure2"
runat="server" Text=""></asp:Label>
            <br />
            <br />
            <asp:Button ID="CmdRafraichir" runat="server"
Text="Rafraichir" onclick="CmdRafraichir_Click" />
        </ContentTemplate>
    </asp:UpdatePanel>
</form>
```

```
' VB .NET

<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

    Date et heure n°1 : <asp:Label ID="LblDateHeure1" runat="server"
Text=""></asp:Label>
    <br />
    <br />

    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            Date et heure n°2 : <asp:Label ID="LblDateHeure2"
runat="server" Text=""></asp:Label>
            <br />
            <br />
            <asp:Button ID="CmdRafraichir" runat="server"
Text="Rafraichir" />
        </ContentTemplate>
    </asp:UpdatePanel>
</form>
```

On obtient alors le résultat suivant :

Date et heure n°1 :

Date et heure n°2 : 28/12/2008 21:51:58

Rafraichir

On peut remarquer que seuls les contrôles contenus dans le contrôle *UpdatePanel*, ont été mis à jour lors de l'exécution de la page côté serveur. Même si le cycle de vie de la page est intégralement exécuté côté serveur, les données renvoyées au client ne contiennent que les informations sur la ou les parties de page à mettre à jour (y compris le ViewState). Voici le flux de données renvoyé par le serveur :

```
269|updatePanel|UpdatePanel1|
                Date et heure n°2 : <span id="LblDateHeure2">28/12/2008
21:51:58</span>
                <br />
                <br />
                <input type="submit" name="CmdRafraichir"
value="Rafraichir" id="CmdRafraichir" />


|164|hiddenField|__VIEWSTATE|/wEPDwUJODQ3ODY5Mjc0D2QWAgIDD2QWBAIDDw8WAh4E
VGV4dAUTMjgvMTIvMjAwOCAyMjoxNTo1MGRkAgUPZBYCZg9kFgICAQ8PFgIfAAUTMjgvMTIvM
jAwOCAyMjoxNTo1MGRkZOiE1zpWhtz9SuEv/4YXgiJ3/0qy|48|hiddenField|__EVENTVAL
IDATION|/wEWAgKUkYeGCALzsoLFCrqoQrEFLEQiRXpzSq6K7WG4TiiX|0|asyncPostBackC
ontrolIDs|||0|postBackControlIDs|||13|updatePanelIDs||tUpdatePanel1|0|chi
ldUpdatePanelIDs|||12|panelsToRefreshIDs||UpdatePanel1|2|asyncPostBackTim
eout||90|12|formAction||Default.aspx|
```

Aussi, côté serveur, étant donné que la page est intégralement exécutée, veuillez bien gérer cette opération de postback particulière. En effet, n'exécutez pas de tâches ou de traitements particuliers, qui n'ont aucune utilité lors de la mise à jour partielle de la page. Pour ce faire, tester la valeur de la propriété *isInAsyncPostBack* du contrôle *ScriptManager*, et la propriété *isInPartialRendering* des contrôles *UpdatePanel*.

6.3.3.2 Définition du contenu

Le contenu du contrôle *UpdatePanel*, peut être défini :

- De manière déclarative, en mode source ou design, lors de la conception d'une page. C'est le cas du premier exemple réalisé ci-dessus.
- De manière impérative :
  - En ajoutant des contrôles à la collection de contrôles, accessibles au travers de la propriété *ContentTemplateContainer* du contrôle *UpdatePanel*.
  - En définissant la propriété *ContentTemplate*, avec tout objet créé à partir d'une classe implémentant l'interface *ITemplate* du Framework .NET. Cette opération peut uniquement être effectuée, si aucun contenu n'a pas été défini de manière déclarative.

```
// C#

UpdatePanel1.ContentTemplate = Page.LoadTemplate("~/UC/UC_Contenu.ascx");
```

```
' VB .NET
UpdatePanel1.ContentTemplate = Page.LoadTemplate("~/UC/UC_Contenu.ascx")
```

6.3.3.3 Mise à jour

Concernant la mise à jour d'un contrôle *UpdatePanel*, les règles de base (par défaut) sont les suivantes :

-
- Tout contrôle situé dans un contrôle *UpdatePanel* réalisant un postback, met à jour :
  - Le contrôle *UpdatePanel* lui-même.
  - Tous les autres contrôles *UpdatePanel* présents dans la page, excepté pour ceux pour lesquels la propriété *UpdateMode* est valorisée à *Conditional*.
- Tout contrôle contenu directement dans la page, réalisant un postback, met à jour la page, et par conséquent les contrôles *UpdatePanel* contenus. Cette règle n'est plus vraie si un trigger (déclencheur) est définie sur un des contrôles *UpdatePanel* de la page. Dans ce cas, on retombe dans le cas précédent. Il existe deux types de triggers :
  - Les triggers asynchrones (*AsyncPostBackTrigger*).
  - Les triggers synchrones (*PostBackTrigger*).

Ces deux triggers possèdent deux propriétés :

- *ControlID* : identifiant du contrôle déclenchant le post back.
- *EventName* : nom de l'évènement du contrôle, à partir duquel le postback sera effectué.

Il est aussi possible de mettre à jour dynamiquement le contrôle *UpdatePanel* côté serveur, via la méthode *Update()*. Mais attention, cette méthode peut lever une exception dans les cas où :

- La propriété *UpdateMode* du contrôle *UpdatePanel* est valorisée à *Always*.
- Si elle est appelée après l'obtention du rendu de la page.

6.4 Le contrôle UpdateProgress

6.4.1 Présentation

La mise à jour de la page peut être potentiellement longue. De plus étant partielle et asynchrone, l'utilisateur ne voit pas qu'une requête a été envoyée au serveur, ce qui peut l'inciter de à envoyer de nouveau une requête, …

Pendant la mise à jour partielle du page, le contrôle *UpdateProgress* permet de prévenir l'utilisateur, qu'une partie d'une page est en cours de mise à jour.

6.4.2 Propriétés

Voici quelques propriétés du contrôle *UpdateProgress*:

Propriété	Description
associatedUpdatePanelId	Identifiant du contrôle *UpdatePanel*, auquel le contrôle *UpdateProgress* est associé
displayAfter	Durée en millisecondes, après laquelle le rendu du contrôle *UpdateProgress* est affiché. Valeur par défaut : 500
dynamicLayout	Valeur booléenne indiquant si le contrôle *UpdateProgress* est rendu dynamiquement dans la page. *True* est la valeur par défaut
progressTemplate	Permet de définir dynamiquement le contenu du contrôle *UpdateProgress*

6.4.3 Association d'un contrôle UpdateProgress à un ou plusieurs contrôles UpdatePanel

La propriété *AssociatedUpdatePanelId* permet de désigner un contrôle *UpdatePanel* particulier. Ainsi, le contrôle *UpdateProgress* apparaîtra lorsque le contrôle *UpdatePanel* associé sera en cours d'exécution. Il disparaîtra automatiquement une fois le rendu contrôle *UpdatePanel* affiché. Si cette propriété n'est pas valorisée, alors le contrôle *UpdateProgress* sera affiché lors de l'exécution de tous les contrôles *UpdatePanel* de la page

6.4.4 Définition du contenu

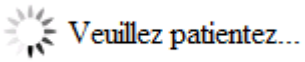
Tout comme le contrôle *UpdatePanel*, Le contrôle *UpdateProgress* peut être mis à jour, soit de manière déclarative (lors de la conception de la page), soit de manière impérative (dynamiquement lors de l'exécution de la page).

6.4.5 Mise en œuvre

Le contrôle *UpdateProgress* est un conteneur de contrôles. Il peut contenir tout type de contrôles ASP .NET. Voici un exemple :

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
    <ProgressTemplate>
        <asp:Image runat="server" ImageUrl="/Images/Attente.gif"
ImageAlign="AbsMiddle" /> Veuillez patientez...
    </ProgressTemplate>
</asp:UpdateProgress>
```

Voici le résultat, qui apparaît automatiquement, lors de la mise à jour partielle d'une page :



6.5 Le contrôle Timer

6.5.1 Présentation

Ce contrôle se comporte de manière similaire à un contrôle *Timer* dans les applications Windows Forms, en exécute des postbacks à intervalle régulier. Il permet ainsi de rafraichir une page Web ou une ou plusieurs parties d'une page Web, à intervalle régulier (suivant sa localisation dans la page, et les triggers définis).

Attention : l'utilisation de ce contrôle peut augmenter le trafic de données entre les clients et le serveur Web.

6.5.2 Propriétés

Ce contrôle possède deux propriétés intéressantes :

Propriété	Description
enabled	Indique si le contrôle *Timer* est actif côté client. *True* est la valeur par défaut.
interval	Intervalle de temps au bout duquel une action est effectuée par le contrôle *Timer* côté serveur. Valeur exprimée en millisecondes. Valeur par défaut : 60 000 (soit 1 minute).

6.5.3 L'évènement Tick

Le contrôle *Timer* possède un évènement *Tick*, qui permet de spécifier les instructions exécutées côté serveur, à chaque de fois que la durée définie dans la propriété *Interval* du contrôle *Timer*, arrive à expiration.

6.5.4 Mise en œuvre

Voici un exemple d'un contrôle *Timer*, réalisant un postback toutes les secondes. En implémentant l'évènement *Tick*, on spécifie les actions à réaliser à chacun de ses postbacks :

```
// C#

<asp:Timer ID="Timer1" runat="server" Enabled="true" Interval="3000"
ontick="Timer1_Tick" />


protected void Timer1_Tick(object sender, EventArgs e)
{
    LblDateHeure3.Text = DateTime.Now.ToString();
}
```

```
// VB .NET


<asp:Timer ID="Timer1" runat="server" Enabled="true" Interval="3000" />

Protected Sub Timer1_Tick(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Timer1.Tick
    LblDateHeure2.Text = DateTime.Now.ToString()
End Sub
```

7 Conclusion

Dans ce chapitre, nous avons mis en évidence les avantages apportés par la technologie Microsoft ASP .NET Ajax, dans le développement d'applications Web. Elle permet de créer des applications plus interactives, offrant de surcroît de meilleurs performances, que des applications Web « non ajaxisées ».