



Dotnet France  
Technologies Sharepoint, SQL Server & .NET

Association Dotnet France

# Consommer des services distants dans les pages ASP .NET Ajax

*Version 1.0*



James RAVAILLE

<http://blogs.dotnet-france.com/jamesr>



# Sommaire

---

1	Introduction.....	3
1.1	Présentation .....	3
1.2	Pré-requis .....	3
2	Architecture pour la communication client / serveur.....	3
2.1	Présentation .....	3
2.2	Côté serveur .....	4
2.3	Côté client.....	4
3	Présentation de l'application à développer .....	5
4	Consommation d'un service WCF .....	7
4.1	Création d'un service WCF .....	7
4.2	Consommation du service WCF .....	9
4.2.1	Enregistrement du service WCF .....	9
4.2.2	Consommation du service WCF dans une page .....	12
5	Consommation d'un service Web Ajax .....	13
5.1	Présentation .....	13
5.2	Création d'un service Web Ajax .....	13
5.3	Consommation du service Web Ajax.....	14
6	Consommation d'une méthode de page.....	17
6.1	Présentation .....	17
6.2	Création d'une méthode de page .....	17
6.3	Consommation de la méthode de page .....	17
6.3.1	Activation de la consommation de méthodes de page.....	17
6.3.2	Consommation de la méthode de page .....	18



## 1 Introduction

### 1.1 Présentation

Dans les applications ASP.NET, la communication entre le client et le serveur, permet d'améliorer nettement les performances et l'interactivité entre les utilisateurs et l'application. Dans ce support de cours, nous allons étudier comment dans un bloc de code JavaScript, consommer des services distants suivants :

- Les services Web
- Les méthodes de page
- Les services WCF

### 1.2 Pré-requis

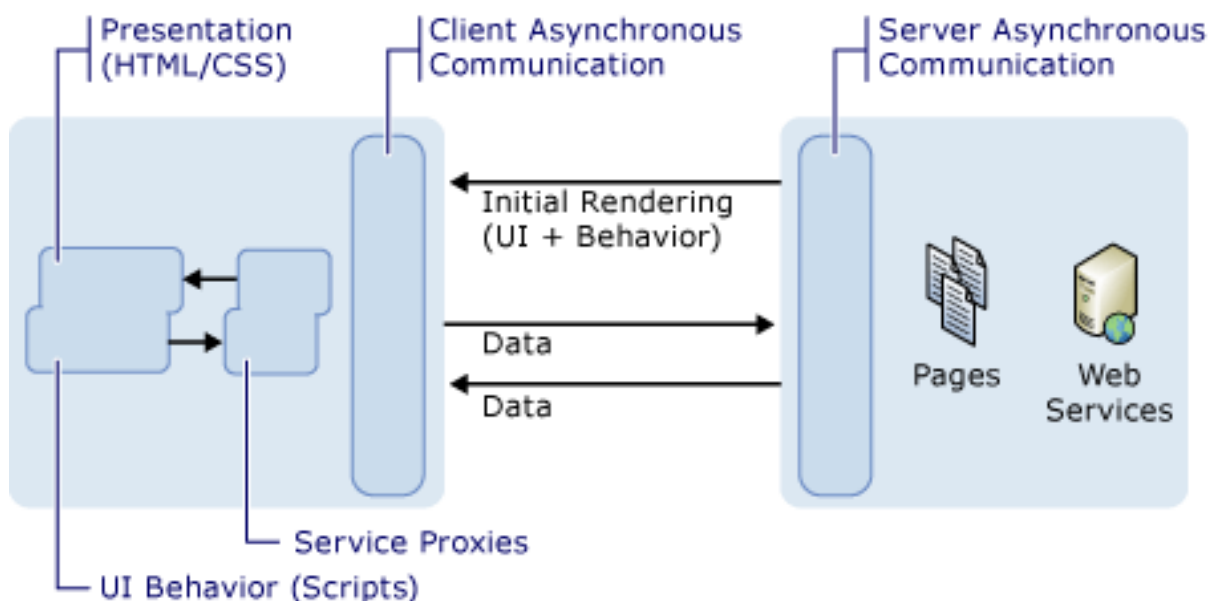
Avant de lire ce cours, vous devez avoir lu les cours suivants :

- Les bases fondamentales de Microsoft ASP.NET Ajax.
- Pour la consommation de services Web : création et consommation de services Web avec le Framework .NET.
- Pour la consommation de services WCF : conception et implémentation de services WCF.

## 2 Architecture pour la communication client / serveur

### 2.1 Présentation

Avec le Framework .NET, Microsoft fournit des composants côté serveur et côté client, permettant à un bloc de code JavaScript de consommer des services distants :





## 2.2 Côté serveur

Côté serveur, Microsoft propose un ensemble de composants, permettant à des clients de consommer des services distants. Pour permettre cette communication, il est nécessaire de définir le module HTTP ScriptModule, dans le fichier de configuration de l'application ASP .NET. Vous pouvez par ailleurs remarquer sa présence dans tout projet ASP .NET, développée avec le Framework .NET 3.5 :

```
// C# et VB .NET

<modules>
  <remove name="ScriptModule" />
  <add name="ScriptModule" preCondition="managedHandler"
type="System.Web.Handlers.ScriptModule, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
</modules>
```

## 2.3 Côté client

Côté client, le Framework .NET génère des classes proxy en JavaScript, permettant de créer des objets qui pourront être sérialisés, de manière à être envoyé à un serveur distant, pour consommer les services distants. Pour enregistrer un service consommable à distance, le contrôle *ScriptManager* un rôle important.



### 3 Présentation de l'application à développer

L'application à développer, permettant de consommer des services distants est très simple :

- Côté serveur, le service doit renvoyer la date et l'heure courante.
- Côté client, la page doit contenir une zone de texte non modifiable (un label), et un simple bouton XHTML, permettant de consommer le service distant.

Cette application servira de base pour les trois exemples qui suivent, qui montrent comment consommer :

- Un service WCF.
- Un service Web.
- Une méthode de page.

Un service WCF ou service Web peut être consommé de toutes pages de l'application, tout comme de l'extérieur de l'application. Une méthode de page est accessible dans le code de la page, et uniquement depuis le code JavaScript de la même page.

Voici le code source :

```
// C# et VB .NET

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Consommation de services distants</title>

  <script language="javascript" type="text/javascript">
    function AfficherDateHeureCourante ()
    {
    }
  </script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

    <input id="CmdAfficherDateHeureCourante" type="button"
value="Afficher la date et l'heure courante"
onclick="AfficherDateHeureCourante();" />

    <br />
    <br />

    Date / heure courante :
    <asp:Label ID="LblDateHeureCourante" runat="server" Text=""
/>
  </form>
</body>
</html>
```

Et lors de l'exécution, on obtient le résultat suivant :



Afficher la date et l'heure courante

Date / heure courante :



## 4 Consommation d'un service WCF

### 4.1 Création d'un service WCF

Dans une application ASP .NET, ajouter un service WCF, nommé *TimeServiceWCF*. La création d'un service WCF crée plusieurs fichiers :

- Dans le répertoire ASP .NET nommé *App\_Code* :
  - o Crée une interface nommée *ITimeServiceWCF*. Cette classe définit et configure les services exposés par le service WCF. Cette interface fait office de contrat :

```
// C#  
  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Runtime.Serialization;  
using System.ServiceModel;  
using System.Text;  
  
// REMARQUE : si vous modifiez le nom d'interface « ITimeServiceWCF »  
// ici, vous devez également mettre à jour la référence à « ITimeServiceWCF »  
// dans Web.config.  
[ServiceContract(Name="TimeServiceWCF", Namespace="ServicesDistantsWCF")]  
public interface ITimeServiceWCF  
{  
    [OperationContract]  
    string GetDateHeureCourante();  
}
```

```
' VB .NET  
  
Imports System.ServiceModel  
  
' REMARQUE : si vous modifiez le nom de classe « ITimeServiceWCF » ici,  
' vous devez également mettre à jour la référence à « ITimeServiceWCF »  
' dans Web.config.  
<ServiceContract(Name:="TimeServiceWCF",  
Namespace:="ServicesDistantsWCF")> _  
Public Interface ITimeServiceWCF  
  
    <OperationContract()> _  
    Function GetDateHeureCourante() As String  
  
End Interface
```

- o Crée une classe nommée *TimeServiceWCF*, qui contient l'implémentation des services :



```
// C#

using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

// REMARQUE : si vous modifiez le nom de classe « TimeServiceWCF » ici,
// vous devez également mettre à jour la référence à « TimeServiceWCF » dans
// Web.config.
public class TimeServiceWCF : ITimeServiceWCF
{
    public string GetDateHeureCourante()
    {
        return DateTime.Now.ToString();
    }
}
```

```
' VB .NET

' REMARQUE : si vous modifiez le nom de classe « TimeServiceWCF » ici,
' vous devez également mettre à jour la référence à « TimeServiceWCF » dans
' Web.config et dans le fichier .svc associé.
Public Class TimeServiceWCF
    Implements ITimeServiceWCF

    Public Function GetDateHeureCourante() As String Implements
ITimeServiceWCF.GetDateHeureCourante
        Return DateTime.Now.ToString()
    End Function
End Class
```

Remarque : le service WCF pourrait avoir des paramètres (bien sûr, sérialisables)...

- Dans le répertoire de l'application ASP .NET, dans lequel le service WCF a été créé, un fichier *TimeServiceWCF.svc* est créé. Il s'agit du point d'accès au service WCF :

```
// C#

<%@ ServiceHost Language="C#" Debug="true" Service="TimeServiceWCF"
CodeBehind="~/App_Code/TimeServiceWCF.cs" %>
```

```
' VB .NET

<%@ ServiceHost Language="VB" Debug="true" Service="TimeServiceWCF"
CodeBehind="~/App_Code/TimeServiceWCF.vb" %>
```



Le fichier de configuration de l'application est modifié, de manière à configurer le service WCF (il est aussi possible d'effectuer ce paramétrage de manière impérative (en code .NET), et aussi pouvoir le consommer :

```
// C# et VB .NET

<system.serviceModel>
  <behaviors>
    <endpointBehaviors>
      <behavior name="webScriptEnablingBehavior">
        <enableWebScript />
      </behavior>
    </endpointBehaviors>
    <serviceBehaviors>
      <behavior name="TimeServiceWCFBehaviors">
        <serviceDebug includeExceptionDetailInFaults="true"/>
        <serviceMetadata httpGetEnabled="true" />
      </behavior>
    </serviceBehaviors>
  </behaviors>

  <services>
    <service behaviorConfiguration="TimeServiceWCFBehaviors"
name="TimeServiceWCF">
      <endpoint behaviorConfiguration="webScriptEnablingBehavior"
        binding="webHttpBinding" bindingConfiguration="default"
contract="ITimeServiceWCF">
        <identity>
          <dns value="localhost" />
        </identity>
      </endpoint>
      <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />
    </service>
  </services>

  <bindings>
    <webHttpBinding>
      <binding name="default" />
    </webHttpBinding>
  </bindings>
</system.serviceModel>
```

## 4.2 Consommation du service WCF

### 4.2.1 Enregistrement du service WCF

Dans un premier temps il est nécessaire d'enregistrer le service comme étant consommable dans un bloc de code JavaScript. Pour ce faire, il faut utiliser le contrôle *ScriptManager* (ou *ScriptManagerProxy*). Il existe deux manières de faire :

- De manière déclarative (codage XHTML) :



```
// C# et VB .NET

<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="~/Services/TimeServiceWCF.svc" />
  </Services>
</asp:ScriptManager>
```

- De manière impérative (code .NET) :

```
// C#

protected void Page_Load(object sender, EventArgs e)
{
    ScriptManager1.Services.Add(new
ServiceReference("~/Services/TimeServiceWCF.svc"));
}
```

```
' VB .NET

Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ScriptManager1.Services.Add(New ServiceReference("~/Services/
TimeServiceWCF.svc"))
End Sub
```

Cette action génère une classe proxy côté client, et sera obtenue les url suivantes :

```
// C#

http://localhost/DF-ConsoServicesDistants-CS/
Services/TimeServiceWCF.svc/js
```

```
' VB .NET

http://localhost/DF-ConsoServicesDistants-VB/
Services/TimeServiceWCF.svc/js
```

Cette classe porte le même nom que le service Web, et expose les méthodes exposées :



```

Type.registerNamespace('ServicesDistantsWCF');
ServicesDistantsWCF.TimeServiceWCF=function() {
ServicesDistantsWCF.TimeServiceWCF.initializeBase(this);
this._timeout = 0;
this._userContext = null;
this._succeeded = null;
this._failed = null;
}
ServicesDistantsWCF.TimeServiceWCF.prototype={
_get_path:function() {
var p = this.get_path();
if (p) return p;
else return
ServicesDistantsWCF.TimeServiceWCF._staticInstance.get_path();},
GetDateHeureCourante:function(succeededCallback, failedCallback,
userContext) {
return this._invoke(this._get_path(),
'GetDateHeureCourante',false,{},succeededCallback,failedCallback,userCont
ext); }}
ServicesDistantsWCF.TimeServiceWCF.registerClass('ServicesDistantsWCF.Tim
eServiceWCF',Sys.Net.WebServiceProxy);
ServicesDistantsWCF.TimeServiceWCF._staticInstance = new
ServicesDistantsWCF.TimeServiceWCF();
ServicesDistantsWCF.TimeServiceWCF.set_path = function(value) {
ServicesDistantsWCF.TimeServiceWCF._staticInstance.set_path(value); }
ServicesDistantsWCF.TimeServiceWCF.get_path = function() { return
ServicesDistantsWCF.TimeServiceWCF._staticInstance.get_path(); }
ServicesDistantsWCF.TimeServiceWCF.set_timeout = function(value) {
ServicesDistantsWCF.TimeServiceWCF._staticInstance.set_timeout(value); }
ServicesDistantsWCF.TimeServiceWCF.get_timeout = function() { return
ServicesDistantsWCF.TimeServiceWCF._staticInstance.get_timeout(); }
ServicesDistantsWCF.TimeServiceWCF.set_defaultUserContext =
function(value) {
ServicesDistantsWCF.TimeServiceWCF._staticInstance.set_defaultUserContext
(value); }
ServicesDistantsWCF.TimeServiceWCF.get_defaultUserContext = function() {
return
ServicesDistantsWCF.TimeServiceWCF._staticInstance.get_defaultUserContext
(); }
ServicesDistantsWCF.TimeServiceWCF.set_defaultSucceededCallback =
function(value) {
ServicesDistantsWCF.TimeServiceWCF._staticInstance.set_defaultSucceededCa
llback(value); }
ServicesDistantsWCF.TimeServiceWCF.get_defaultSucceededCallback =
function() { return
ServicesDistantsWCF.TimeServiceWCF._staticInstance.get_defaultSucceededCa
llback(); }
ServicesDistantsWCF.TimeServiceWCF.set_defaultFailedCallback =
function(value) {
ServicesDistantsWCF.TimeServiceWCF._staticInstance.set_defaultFailedCallb
ack(value); }
ServicesDistantsWCF.TimeServiceWCF.get_defaultFailedCallback = function()
{ return
ServicesDistantsWCF.TimeServiceWCF._staticInstance.get_defaultFailedCallb
ack(); }
ServicesDistantsWCF.TimeServiceWCF.set_path("/DF-ConsoServicesDistants-
VB/Services/TimeServiceWCF.svc");
ServicesDistantsWCF.TimeServiceWCF.GetDateHeureCourante=
function(onSuccess,onFailed,userContext)
{ServicesDistantsWCF.TimeServiceWCF._staticInstance.GetDateHeureCourante(
onSuccess,onFailed,userContext); }

```



#### 4.2.2 Consommation du service WCF dans une page

Puis, dans un bloc de code JavaScript de l'application ASP .NET, on peut créer trois méthode :

- Une méthode qui devra être exécutée si l'appel au service WCF réussit :

```
function TraiterReussiteAppelServiceDistant(aResultat, aContexte,
aNomMethode) {
    $get("LblDateHeureCourante").innerHTML = aResultat;
}
```

- Une méthode qui devra être exécutée si l'appel au service WCF échoue :

```
function TraiterEchecAppelServiceDistant(aResultat, aContexte,
aNomMethode) {
    $get("LblDateHeureCourante").innerHTML =
        "Une erreur est survenue lors de l'exécution du service WCF '"
        + aNomMethode + "' : " + aResultat.get_message();
}
```

- Une méthode exécutant l'appel au service WCF, qui pourra être abonné à un évènement côté client d'un contrôle XHTML :

```
function AfficherDateHeureCourante() {
    ServicesDistantsWCF.TimeServiceWCF.GetDateHeureCourante(TraiterReussiteAp
    pelServiceDistant, TraiterEchecAppelServiceDistant, null);
}
```

Voici quelques précisions sur la signature de la méthode *GetDateHeureCourante* :

- Les premiers paramètres correspondent aux paramètres de la méthode de page (la méthode du service WCF n'en ayant pas, cette méthode n'en spécifie pas).
- Le paramètre suivant correspond au nom de la fonction JavaScript automatiquement appelée, si l'appel de la méthode de page réussit.
- Le paramètre suivant correspond au nom de la fonction JavaScript automatiquement appelée, si l'appel de la méthode de page échoue.
- Le paramètre suivant correspond à un contexte de données, fourni à la méthode de réussite ou d'échec.

En exécutant notre page ASP .NET, on obtient le résultat suivant :

Afficher la date et l'heure courante

Date / heure courante : 30/12/2008 09:16:59



## 5 Consommation d'un service Web Ajax

### 5.1 Présentation

Un service Web Ajax se conçoit comme à un service Web classique. Toutefois, il est obligatoire de définir le service Web avec la métadonnée *System.Web.Script.Services.ScriptService*. Pour pouvoir « configurer » l'exposition d'une méthode Web au code JavaScript d'une page de l'application, il faut utiliser l'attribut *System.Web.Script.Services.ScriptMethod* (cet attribut est optionnel).

### 5.2 Création d'un service Web Ajax

Dans votre application .NET, créons un service Web nommé *TimeService.asmx*. La création d'un service Web crée plusieurs fichiers :

- Dans le répertoire ASP.NET nommé *App\_code*, une classe nommée *TimeService* est créée. Cette classe contient l'implémentation des méthodes Web du Service Web. Implémentons alors la méthode Web *GetDateHeureCourante()*, qui retourne la date et l'heure courante :

```
// C#  
  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Web;  
using System.Web.Services;  
  
/// <summary>  
/// Description résumée de TimeService  
/// </summary>  
[WebService(Namespace = "http://tempuri.org/")]  
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]  
[System.Web.Script.Services.ScriptService()]  
public class TimeService : System.Web.Services.WebService {  
  
    public TimeService ()  
    {  
        [WebMethod]  
        [System.Web.Script.Services.ScriptMethod()]  
        public string GetDateHeureCourante () {  
            return DateTime.Now.ToString();  
        }  
    }  
}
```



```

' VB .NET

Imports System.Web
Imports System.Web.Services
Imports System.Web.Services.Protocols

<System.Web.Services.WebService(Namespace:="http://tempuri.org/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<Global.Microsoft.VisualBasic.CompilerServices.DesignerGenerated()> _
<System.Web.Script.Services.ScriptService()> _
Public Class TimeService
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    <Script.Services.ScriptMethod()> _
    Public Function GetDateHeureCourante() As String
        Return DateTime.Now.ToString()
    End Function
End Class
  
```

La métadonnée *ScriptMethod* défini sur la méthode Web est optionnelle. Elle permet de définir un comportement particulier de la méthode Web, au travers de propriétés qu'elle propose :

Propriété	Description
<i>responseFormat</i>	Spécifie si la réponse doit être sérialisée au format JSON (par défaut) ou XML
<i>useHttpGet</i>	Indique si le verbe HTTP Get peut être utilisé pour invoquer la méthode (false par défaut)
<i>xmlSerializeString</i>	Indique si le type de retour est sérialisable au format XML (false par défaut). Ignorée si la réponse est au format JSON

Remarque : le service Web Ajax pourrait aussi avoir des paramètres (bien sûr, sérialisables)...

### 5.3 Consommation du service Web Ajax

Comme un service WCF, il est nécessaire d'enregistrer le service Web, via le contrôle *ScriptManager* ou *ScriptManagerProxy*. Il existe, là aussi, deux manières de faire :

- De manière déclarative (codage XHTML) :



```
// C# et VB .NET

<asp:ScriptManager ID="ScriptManager1" runat="server"
EnablePageMethods="true">
  <Services>
    <asp:ServiceReference Path="~/Services/TimeService.asmx" />
  </Services>
</asp:ScriptManager>
```

- De manière impérative (code .NET) :

```
// C#

protected void Page_Load(object sender, EventArgs e)
{
    ScriptManager1.Services.Add(new
ServiceReference("~/Services/TimeService.asmx"));
}
```

```
' VB .NET

Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    ScriptManager1.Services.Add(New
ServiceReference("~/Services/TimeService.asmx"))
End Sub
```

Cette action génère une classe proxy côté client. Cette classe porte le même nom que le service Web, et expose les méthodes Web qu'il possède (les méthodes portant les mêmes noms...). Ainsi, pour le consommer :

```
' C# et VB.NET

function AfficherDateHeureCourante() {
    TimeService.GetDateHeureCourante(TraiterReussiteAppelServiceDistant,
    TraiterEchecAppelServiceDistant, null);
}
```

Les fonctions JavaScript de traitement de la réussite et de l'échec de l'appel au service Web Ajax, sont, dans notre cas, sensiblement identiques à celles vues précédemment, avec la consommation du service WCF :



```
' C# et VB.NET

function TraiterEchecAppelServiceDistant(aResultat, aContexte,
aNomMethode) {
    $get("LblDateHeureCourante").innerHTML =
        "Une erreur est survenue lors de l'exécution du service WCF
'" + aNomMethode + "' : " + aResultat.get_message();
}

function TraiterReussiteAppelServiceDistant(aResultat, aContexte,
aNomMethode) {
    $get("LblDateHeureCourante").innerHTML = aResultat;
}
}
```

Voici le résultat de l'exécution du Service Web Ajax :

Afficher la date et l'heure courante

Date / heure courante : 28/12/2008 00:35:15



## 6 Consommation d'une méthode de page

### 6.1 Présentation

Une méthode de page est une méthode contenue dans une classe code-behind d'une page ASP .NET, qui possède les caractéristiques suivantes :

- Elle est publique ;
- Elle est statique ;
- Elle est définie avec la métadonnée `System.Web.Services.WebMethod`. En effet, la méthode de page est exposée comme une méthode Web d'un service Web ;
- Elle peut avoir des paramètres d'entrée.

### 6.2 Création d'une méthode de page

Voici un exemple d'une méthode de page, qui retourne la date et l'heure courante :

```
// C#  
  
[System.Web.Services.WebMethod()]  
public static string GetDateHeureCourante()  
{  
    return DateTime.Now.ToString();  
}
```

```
' VB.NET  
  
<System.Web.Services.WebMethod()> _  
Public Shared Function GetDateHeureCourante() As String  
    Return DateTime.Now.ToString()  
End Function
```

### 6.3 Consommation de la méthode de page

#### 6.3.1 Activation de la consommation de méthodes de page

Pour consommer une méthode de page, il est nécessaire d'autoriser cette action sur le contrôle *ScriptManager*, en valorisant sa propriété `EnablePageMethods` à `true`. Attention, elle ne peut être réalisée sur le contrôle *ScriptManagerProxy* :

```
' C# et VB.NET  
  
<asp:ScriptManager ID="ScriptManager1" runat="server"  
EnablePageMethods="true">  
</asp:ScriptManager>
```

Cette action permet d'exposer les méthodes de page au code JavaScript de la même page, via l'objet *PageMethod*.



### 6.3.2 Consommation de la méthode de page

Dans la fonction JavaScript *AfficherDateHeureCourante*, utiliser l'objet *PageMethods*, qui expose automatiquement les méthodes de page, définies dans la page ASP .NET. Ces méthodes ont le même nom que les méthodes de page :

```
' C# et VB.NET

<script language="javascript" type="text/javascript">
  function AfficherDateHeureCourante() {
    PageMethods.GetDateHeureCourante(TraiterReussiteAppelServiceDistant,
      TraiterEchecAppelServiceDistant, null);
  }

  function TraiterEchecAppelServiceDistant(aResultat, aContexte,
aNomMethode) {
    $get("LblDateHeureCourante").innerHTML =
      "Une erreur est survenue lors de l'exécution de la méthode '"
      + aNomMethode + "' : " + aResultat.get_message();
  }

  function TraiterReussiteAppelServiceDistant(aResultat, aContexte,
aNomMethode) {
    $get("LblDateHeureCourante").innerHTML = aResultat;
  }
</script>
```

Voilà le résultat de l'exécution :

Afficher la date et l'heure courante

Date / heure courante : 27/12/2008 23:56:47