



eBook Gratuit

APPRENEZ ASP.NET

eBook gratuit non affilié créé à partir des
contributeurs de Stack Overflow.

#asp.net

Table des matières

À propos	1
Chapitre 1: Démarrer avec ASP.NET	2
Remarques	2
Exemples	2
Installation ou configuration	2
ASP.NET Vue d'ensemble	2
Bonjour tout le monde avec OWIN	3
Introduction simple d'ASP.NET	4
Chapitre 2: Afficher l'état	5
Introduction	5
Syntaxe	5
Exemples	5
Exemple	5
Chapitre 3: Asp Web Forms Identity	7
Exemples	7
Commencer	7
Chapitre 4: ASP.NET - Contrôles de base	9
Syntaxe	9
Exemples	9
Boîtes de texte et étiquettes	9
Cases à cocher et boutons radio	10
Commandes de liste	11
Liste des boutons radio et liste des cases à cocher	12
Listes à puces et listes numérotées	13
Contrôle HyperLink	13
Contrôle d'image	14
Chapitre 5: ASP.NET - Contrôles utilisateur	15
Introduction	15
Exemples	15
Introduction des contrôles utilisateur	15

Création d'une instance de contrôle utilisateur par programme.....	16
Ajout de propriétés personnalisées pour le contrôle utilisateur.....	17
Chapitre 6: ASP.NET - Etat de gestion.....	18
Exemples.....	18
Afficher l'état.....	18
Chapitre 7: ASP.NET - Validateurs.....	20
Syntaxe.....	20
Exemples.....	20
Contrôles de validation.....	20
Contrôle RequiredFieldValidator.....	20
Contrôle de RangeValidator.....	21
CompareValidator Control.....	21
RegularExpressionValidator.....	22
Résumé de validation.....	23
Groupes de validation.....	24
Chapitre 8: Contrôles Asp.net Ajax.....	27
Exemples.....	27
FileUpload Ajax Toolkit Control.....	27
Chapitre 9: Cycle de vie de la page.....	29
Exemples.....	29
Événements du cycle de vie.....	29
Exemple de code.....	30
Chapitre 10: Délégation d'événement.....	34
Syntaxe.....	34
Remarques.....	34
Exemples.....	34
Délégation d'événement du contrôle utilisateur à aspx.....	34
Chapitre 11: Directives.....	37
Exemples.....	37
La directive d'application.....	37
La directive de contrôle.....	37
La directive sur les outils.....	38

La directive maîtrise	38
La directive sur les importations	38
La directive MasterType	38
La directive page	39
La directive OutputCache	40
Chapitre 12: Etat de session	41
Syntaxe	41
Remarques	41
Exemples	41
Utilisation de l'objet Session pour stocker des valeurs	41
Utilisation d'un magasin de sessions SQL	42
Utilisation d'un magasin de sessions Amazon DynamoDB	42
Chapitre 13: Expressions	44
Exemples	44
Valeur de App.Config	44
Expression évaluée	44
Bloc de code dans le balisage ASP	44
Chapitre 14: Gestion de session	45
Exemples	45
Avantage et désavantage de l'état de session, types de session	45
Chapitre 15: Gestion des événements	46
Syntaxe	46
Paramètres	46
Exemples	46
Événements d'application et de session	46
Événements de page et de contrôle	46
Événements par défaut	47
Chapitre 16: GridView	50
Exemples	50
Liaison de données	50
Reliure manuelle	50
DataSourceControl	50

Colonnes.....	50
GridView fortement typé.....	51
Événement de commande de manipulation.....	52
Pagination.....	53
ObjectDataSource.....	54
Reliure manuelle.....	54
Cliquez sur Mettre à jour la grille en ligne.....	55
Chapitre 17: httpHandlers.....	58
Exemples.....	58
Utiliser un httpHandler (.ashx) pour télécharger un fichier à partir d'un emplacement spéc.....	58
Chapitre 18: Katana.....	60
Introduction.....	60
Exemples.....	60
Exemple.....	60
Chapitre 19: Liaison de données.....	62
Exemples.....	62
Source de données SQL.....	62
Récupération des données.....	62
Utilisation de base.....	63
Source de données d'objet.....	63
Chapitre 20: Liste de données.....	65
Syntaxe.....	65
Exemples.....	65
Liaison de données dans asp.net.....	65
Chapitre 21: Méthodes de page.....	67
Paramètres.....	67
Remarques.....	67
Plus d'un paramètre.....	67
Valeur de retour.....	67
Exemples.....	67
Comment l'appeler.....	67

Chapitre 22: Middleware	69
Paramètres	69
Remarques	69
Exemples	69
Affiche le chemin de la requête et le temps nécessaire pour le traiter	69
Chapitre 23: Mise en cache ASP.NET	71
Exemples	71
Cache de données	71
Chapitre 24: Planificateur DayPilot	73
Paramètres	73
Remarques	73
Exemples	73
Informations de base	73
Déclaration	73
Chapitre 25: Rechercher le contrôle par ID	75
Syntaxe	75
Remarques	75
Exemples	75
Accéder au contrôle TextBox dans la page aspx	75
Trouvez un contrôle dans un GridView, un répéteur, une liste, etc	75
Chapitre 26: Répéteur	76
Exemples	76
Utilisation de base	76
Chapitre 27: ScriptManager	77
Introduction	77
Syntaxe	77
Exemples	77
Travailler avec ScriptManager	77
Chapitre 28: sections web.config> system.webServer / httpErrors & system.web / customError	79
Introduction	79
Exemples	79

Quelle est la différence entre customErrors et httpErrors?.....	79
Chapitre 29: UpdatePanel.....	80
Introduction.....	80
Syntaxe.....	80
Remarques.....	80
Exemples.....	80
Mise à jour de l'exemple de panneau.....	80
Chapitre 30: WebForms.....	82
Syntaxe.....	82
Remarques.....	82
Exemples.....	82
Utilisation d'un répéteur pour créer une table HTML.....	82
Grouper dans ListView.....	83
Exemple.....	85
Lien hypertexte.....	85
Chapitre 31: WebService sans Visual Studio.....	87
Introduction.....	87
Remarques.....	87
Exemples.....	87
Calculatrice WebService.....	87
Crédits.....	88

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: [asp-net](#)

It is an unofficial and free ASP.NET ebook created for educational purposes. All the content is extracted from [Stack Overflow Documentation](#), which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official ASP.NET.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec ASP.NET

Remarques

ASP.NET est un ensemble de technologies du .NET Framework destinées au développement d'applications Web. Ces technologies consistent en:

- WebForms: Plate-forme de développement de style RAD utilisant des contrôles Web.
- MVC: plate-forme de développement Model View Controller.
- SignalR: plate-forme de messagerie en temps réel pour la messagerie client / serveur.
- Razor: un langage de balisage frontal avec lequel vous pouvez incorporer des commandes côté serveur.
- WebAPI: plate-forme de création d'applications de style API REST.

Exemples

Installation ou configuration

Par défaut, toutes les bibliothèques requises pour la génération d'applications ASP.NET sont incluses lors de l'installation de Visual Studio. Si une version plus récente d'ASP.NET est publiée et n'est pas incluse avec Visual Studio, vous pouvez télécharger la bibliothèque SDK appropriée de Microsoft, qui inclura toutes les bibliothèques nécessaires pour cette version.

De même, le système d'exploitation Windows est pré-installé avec une version plus récente d'ASP.NET et est automatiquement enregistré avec IIS pour la configuration et l'exécution. De même, si une version plus récente d'ASP.NET devient disponible, vous pouvez installer le SDK correspondant à la version dont vous avez besoin, puis utiliser l'outil `aspnet_regiis` pour enregistrer l' `aspnet_regiis` avec IIS.

Il convient également de noter que pour les déploiements de serveurs, il existe également un package ASP.NET SDK Redistributable. Cette version est une version simplifiée du SDK, avec juste les bibliothèques essentielles et ne possède pas les outils et intégrations avec Visual Studio.

ASP.NET Vue d'ensemble

ASP.NET est un modèle de développement Web unifié qui inclut les services nécessaires à la création d'applications Web d'entreprise avec un minimum de codage. ASP.NET fait partie du .NET Framework et, lors du codage des applications ASP.NET, vous avez accès aux classes du .NET Framework.

Vous pouvez coder vos applications dans n'importe quel langage compatible avec le Common Language Runtime (CLR), notamment Microsoft Visual Basic, C #, JScript .NET et J #. Ces langages vous permettent de développer des applications ASP.NET bénéficiant du Common Language Runtime, de la sécurité de type, de l'héritage, etc.

ASP.NET comprend:

- Un cadre de page et de contrôles
- Le compilateur ASP.NET
- Infrastructure de sécurité
- Installations de gestion d'Etat
- Configuration de l'application
- Surveillance de la santé et caractéristiques de performance
- Prise en charge du débogage
- Un framework de services Web XML
- Environnement d'hébergement extensible et gestion du cycle de vie des applications
- Un environnement de conception extensible

Bonjour tout le monde avec OWIN

Utilisez le gestionnaire de paquets pour installer Microsoft.Owin.SelfHost

```
install-package Microsoft.Owin.SelfHost
```

Code pour une application web HelloWorld minimale exécutée à partir d'une fenêtre de la console:

```
namespace HelloOwin
{
    using System;
    using Owin;

    class Program
    {
        static readonly string baseUrl = "http://localhost:8080";

        static void Main(string[] args)
        {
            using (Microsoft.Owin.Hosting.WebApp.Start<Startup>(baseUrl))
            {
                Console.WriteLine("Prease any key to quit.");
                Console.ReadKey();
            }
        }
    }

    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.Run(ctx =>
            {
                return ctx.Response.WriteAsync("Hello World");
            });
        }
    }
}
```


Introduction simple d'ASP.NET

Asp.net est un framework d'applications Web développé par Microsoft pour créer des applications Web dynamiques et des WebServices.

Asp.net est essentiellement un sous-ensemble du framework .NET plus large. Un framework n'est rien d'autre qu'une collection de classes.

Dans .NET Framework, vous pouvez créer une application console. Application Web, Application de fenêtre, Application mobile. Ainsi, pour l'application Web ASP.net est utilisé.

ASP.NET est le successeur de ASP classique (Active Server Page.)

Qu'est-ce qu'une application Web?

Une application Web est une application accessible aux utilisateurs via un navigateur Web, par exemple:

- Microsoft Internet Explorer.
- Google Chrome
- Mozilla FireFox
- Safari d'Apple

Lire Démarrer avec ASP.NET en ligne: <https://riptutorial.com/fr/asp-net/topic/836/demarrer-avec-asp-net>

Chapitre 2: Afficher l'état

Introduction

View State est la méthode permettant de préserver la valeur de la page et les contrôles entre les allers-retours. C'est une technique de gestion de niveau page. L'état d'affichage est activé par défaut et sérialise normalement les données dans chaque contrôle de la page, même s'il est réellement utilisé lors d'un post-retour.

Syntaxe

- ViewState ["NameofViewstate"] = "Valeur";

Exemples

Exemple

ASPX

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>ViewState</title>
</head>
<body>
<form id="form1" runat="server">
<asp:TextBox runat="server" id="NameField" />
<asp:Button runat="server" id="SubmitForm" onclick="SubmitForm_Click" text="Submit
& set name" />
<asp:Button runat="server" id="RefreshPage" text="Just submit" />
<br /><br />
Name retrieved from ViewState: <asp:Label runat="server" id="NameLabel" />
</form>
</body>
</html>
```

Code derrière

```
using System;
using System.Data;
using System.Web;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (ViewState["NameOfUser"] != null)
```



```
        NameLabel.Text = ViewState["NameOfUser"].ToString();
    else
        NameLabel.Text = "Not set yet...";
}

protected void SubmitForm_Click(object sender, EventArgs e)
{
    ViewState["NameOfUser"] = NameField.Text;
    NameLabel.Text = NameField.Text;
}
}
```

Lire Afficher l'état en ligne: <https://riptutorial.com/fr/asp-net/topic/8234/afficher-l-etat>

Chapitre 3: Asp Web Forms Identity

Exemples

Commencer

Commencer

Installez les paquets NuGet:

1. **Microsoft.AspNet.Identity.EntityFramework**
2. **Microsoft.AspNet.Identity.Core**
3. **Microsoft.AspNet.Identity.OWIN**

Enregistrer une action - Contrôleur de compte

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Register(RegisterViewModel model)
{
    if (ModelState.IsValid)
    {
        var user = new ApplicationUser() { UserName = model.UserName };
        var result = await UserManager.CreateAsync(user, model.Password);
        if (result.Succeeded)
        {
            await SignInAsync(user, isPersistent: false);
            return RedirectToAction("Index", "Home");
        }
        else
        {
            AddErrors(result);
        }
    }

    // If we got this far, something failed, redisplay form
    return View(model);
}
```

Action de connexion - Méthode SignInAsync

```
private async Task SignInAsync(ApplicationUser user, bool isPersistent)
{
    AuthenticationManager.SignOut(DefaultAuthenticationTypes.ExternalCookie);

    var identity = await UserManager.CreateIdentityAsync(
        user, DefaultAuthenticationTypes.ApplicationCookie);

    AuthenticationManager.SignIn(
        new AuthenticationProperties() {
```



```
        IsPersistent = isPersistent
    }, identity);
}
```

Se déconnecter

```
// POST: /Account/LogOff
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult LogOff()
{
    AuthenticationManager.SignOut();
    return RedirectToAction("Index", "Home");
}
```

Lire Asp Web Forms Identity en ligne: <https://riptutorial.com/fr/asp-net/topic/9146/asp-web-forms-identity>

Chapitre 4: ASP.NET - Contrôles de base

Syntaxe

- `<asp: Button ID = "Button1" runat = "serveur" onclick = "Button1_Click" Text = "Cliquez" />`
`<asp: TextBox ID = "txtstate" runat = "serveur">`
- `</ asp: TextBox> <asp: CheckBox ID = "chkoption" runat = "Serveur"> </ asp: CheckBox>`
`<asp: ID RadioButton = "rdboption" runat = "Serveur"> </ asp: RadioButton>`
- `<asp: ListBox ID = "ListBox1" runat = "serveur" AutoPostBack = "True"`
`OnSelectedIndexChanged = "ListBox1_SelectedIndexChanged"> </ asp: ListBox>`
- `<asp: DropDownList ID = "DropDownList1" runat = "serveur" AutoPostBack = "True"`
`OnSelectedIndexChanged = "DropDownList1_SelectedIndexChanged"> </ asp:`
`DropDownList>`
- `<asp: ID RadioButtonList = "RadioButtonList1" runat = "serveur" AutoPostBack = "True"`
`OnSelectedIndexChanged = "RadioButtonList1_SelectedIndexChanged"> </ asp:`
`RadioButtonList>`
- `<asp: CheckBoxList ID = "CheckBoxList1" runat = "serveur" AutoPostBack = "True"`
`OnSelectedIndexChanged = "CheckBoxList1_SelectedIndexChanged"> </ asp:`
`CheckBoxList>`
- `<asp: BulletedList ID = "BulletedList1" runat = "serveur"> </ asp: BulletedList>`
- `<asp: HyperLink ID = "HyperLink1" runat = "serveur"> HyperLink </ asp: HyperLink> <asp:`
`Image ID = "Image1" runat = "serveur">`

Exemples

Boîtes de texte et étiquettes

Les contrôles de zone de texte sont généralement utilisés pour accepter les entrées de l'utilisateur. Un contrôle de zone de texte peut accepter une ou plusieurs lignes de texte en fonction des paramètres de l'attribut `TextMode`.

Les contrôles d'étiquettes permettent d'afficher facilement le texte qui peut être modifié d'une exécution d'une page à l'autre. Si vous souhaitez afficher un texte qui ne change pas, vous utilisez le texte littéral.

Syntaxe de base du contrôle de texte:

```
<asp:TextBox ID="txtstate" runat="server" ></asp:TextBox>
```

Propriétés communes de la zone de texte et des étiquettes:

Propriétés	La description
Mode texte	Spécifie le type de zone de texte. <code>SingleLine</code> crée une zone de texte standard, <code>MultiLine</code> crée une zone de texte qui accepte plusieurs lignes de

Propriétés	La description
	texte et le mot de passe masque les caractères entrés. La valeur par défaut est SingleLine.
Texte	Le contenu textuel de la zone de texte.
Longueur maximale	Nombre maximal de caractères pouvant être saisis dans la zone de texte.
Emballage	Il détermine si le texte retourne automatiquement ou non pour la zone de texte multiligne. le défaut est vrai.
Lecture seulement	Détermine si l'utilisateur peut modifier le texte dans la boîte. la valeur par défaut est false, c.-à-d. que l'utilisateur peut modifier le texte.
Colonnes	La largeur de la zone de texte en caractères. La largeur réelle est déterminée en fonction de la police utilisée pour la saisie de texte.
Des rangées	La hauteur d'une zone de texte multiligne en lignes. La valeur par défaut est 0, signifie une zone de texte à une seule ligne.

L'attribut le plus souvent utilisé pour un contrôle d'étiquette est «Texte», ce qui implique le texte affiché sur l'étiquette.

Cases à cocher et boutons radio

Une case à cocher affiche une option unique que l'utilisateur peut cocher ou décocher et les boutons radio présentent un groupe d'options à partir duquel l'utilisateur peut sélectionner une seule option.

Pour créer un groupe de boutons radio, spécifiez le même nom pour l'attribut GroupName de chaque bouton radio du groupe. Si plusieurs groupes sont requis dans un seul formulaire, spécifiez un nom de groupe différent pour chaque groupe.

Si vous souhaitez que la case à cocher ou le bouton radio soit sélectionné lors de l'affichage initial du formulaire, définissez son attribut Checked sur true. Si l'attribut Checked est défini sur true pour plusieurs boutons radio d'un groupe, seul le dernier est considéré comme true.

Syntaxe de base de la case à cocher:

```
<asp:CheckBox ID= "chkoption" runat= "Server"> </asp:CheckBox>
```

Syntaxe de base du bouton radio:

```
<asp:RadioButton ID= "rdboption" runat= "Server"> </asp: RadioButton>
```

Propriétés communes des cases à cocher et des boutons radio:

Propriétés	La description
Texte	Le texte affiché à côté de la case à cocher ou du bouton radio.
Vérifié	Spécifie s'il est sélectionné ou non, la valeur par défaut est false.
Nom de groupe	Nom du groupe auquel le contrôle appartient.

Commandes de liste

ASP.NET fournit les commandes suivantes

- La liste déroulante
- Zone de liste
- Liste de boutons radio
- Liste de case à cocher
- Liste à puces

Ces contrôles permettent à un utilisateur de choisir parmi un ou plusieurs éléments de la liste. Les zones de liste et les listes déroulantes contiennent un ou plusieurs éléments de liste. Ces listes peuvent être chargées soit par code, soit par l'éditeur ListItemCollection.

Syntaxe de base du contrôle de la zone de liste:

```
<asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
</asp:ListBox>
```

Syntaxe de base du contrôle de liste déroulante:

```
<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
</asp:DropDownList>
```

Propriétés communes des listes et listes déroulantes:

Propriétés	La description
Articles	La collection d'objets ListItem qui représente les éléments du contrôle. Cette propriété renvoie un objet de type ListItemCollection.
Des rangées	Spécifie le nombre d'éléments affichés dans la boîte. Si la liste actuelle contient plus de lignes que celles affichées, une barre de défilement est ajoutée.
SelectedIndex	L'index de l'élément actuellement sélectionné. Si plusieurs éléments sont sélectionnés, l'index du premier élément sélectionné. Si aucun élément n'est sélectionné, la valeur de cette propriété est -1.
Valeur	La valeur de l'élément actuellement sélectionné. Si plusieurs éléments sont

Propriétés	La description
sélectionnée	sélectionnés, la valeur du premier élément sélectionné. Si aucun élément n'est sélectionné, la valeur de cette propriété est une chaîne vide ("").
Mode de selection	Indique si une zone de liste permet des sélections uniques ou des sélections multiples.

Propriétés communes des objets de chaque élément de la liste:

Propriétés	La description
Texte	Le texte affiché pour l'article.
Choisi	Valeur de chaîne associée à l'élément.
Valeur	Indique si l'élément est sélectionné.

Il est important de noter que:

- Pour utiliser les éléments dans une liste déroulante ou une zone de liste, utilisez la propriété Items du contrôle. Cette propriété renvoie un objet ListItemCollection qui contient tous les éléments de la liste.
- L'événement SelectedIndexChanged est déclenché lorsque l'utilisateur sélectionne un autre élément dans une liste déroulante ou une zone de liste.

Liste des boutons radio et liste des cases à cocher

Une liste de boutons radio présente une liste d'options mutuellement exclusives. Une liste de cases à cocher présente une liste d'options indépendantes. Ces contrôles contiennent une collection d'objets ListItem pouvant être référencés via la propriété Items du contrôle.

Syntaxe de base de la liste des boutons radio:

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True"
    OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
</asp:RadioButtonList>
```

Syntaxe de base de la liste des cases à cocher:

```
<asp:CheckBoxList ID="CheckBoxList1" runat="server" AutoPostBack="True"
    OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
</asp:CheckBoxList>
```

Propriétés communes des listes de cases à cocher et de bouton radio:

Propriétés	La description
RepeatLayout	Cet attribut spécifie si les balises de table ou le flux HTML normal à utiliser

Propriétés	La description
	lors du formatage de la liste lors du rendu. La valeur par défaut est Table.
RepeatDirection	Il spécifie la direction dans laquelle les contrôles doivent être répétés. Les valeurs disponibles sont Horizontal et Vertical. La valeur par défaut est verticale.
RepeatColumns	Il spécifie le nombre de colonnes à utiliser lors de la répétition des contrôles. la valeur par défaut est 0.

Listes à puces et listes numérotées

Le contrôle de liste à puces crée des listes à puces ou des listes numérotées. Ces contrôles contiennent une collection d'objets ListItem pouvant être référencés via la propriété Items du contrôle.

Syntaxe de base d'une liste à puces:

```
<asp:BulletedList ID="BulletedList1" runat="server">
</asp:BulletedList>
```

Propriétés communes de la liste à puces:

Propriétés	La description
BulletStyle	Cette propriété spécifie le style et l'apparence des puces ou des nombres.
RepeatDirection	Il spécifie la direction dans laquelle les contrôles doivent être répétés. Les valeurs disponibles sont Horizontal et Vertical. La valeur par défaut est verticale.
RepeatColumns	Il spécifie le nombre de colonnes à utiliser lors de la répétition des contrôles. la valeur par défaut est 0.

Contrôle HyperLink

Le contrôle HyperLink ressemble à l'élément HTML.

Syntaxe de base pour un contrôle de lien hypertexte:

```
<asp:HyperLink ID="HyperLink1" runat="server">
    HyperLink
</asp:HyperLink>
```

Il possède les propriétés importantes suivantes:

Propriétés	La description
URL de l'image	Chemin de l'image à afficher par le contrôle.
NavigateUrl	URL du lien cible.
Texte	Le texte à afficher comme lien.
Cible	La fenêtre ou le cadre qui charge la page liée.

Contrôle d'image

Le contrôle d'image est utilisé pour afficher des images sur la page Web ou un autre texte, si l'image n'est pas disponible.

Syntaxe de base pour un contrôle d'image:

```
<asp:Image ID="Image1" runat="server">
```

Il possède les propriétés importantes suivantes:

Propriétés	La description
Texte alternatif	Texte alternatif à afficher en l'absence de l'image.
ImageAlign	Options d'alignement pour le contrôle.
URL de l'image	Chemin de l'image à afficher par le contrôle.

Lire ASP.NET - Contrôles de base en ligne: <https://riptutorial.com/fr/asp-net/topic/6444/asp-net---controles-de-base>

Chapitre 5: ASP.NET - Contrôles utilisateur

Introduction

Les contrôles utilisateur sont des conteneurs pouvant être remplis avec des contrôles HTML et des contrôles serveur avec code-behind de la même manière que la page ASPX. Ils sont traités comme des unités plus petites réutilisables d'une page, de sorte qu'ils ne peuvent pas fonctionner sous forme de pages autonomes et ne doivent pas avoir **html**, le **corps** ou la **forme** des éléments HTML en eux.

Exemples

Introduction des contrôles utilisateur

Les contrôles utilisateur permettent de réutiliser les pages ASP.NET, de la même manière que les pages maîtres. Au lieu de partager la mise en page de base, les contrôles utilisateur partagent un groupe de contrôles serveur HTML / ASP.NET intégrés ou une disposition de formulaire spécifique, par exemple l'envoi de commentaires ou les notes invité.

Un contrôle utilisateur peut contenir des contrôles HTML et des contrôles serveur ASP.NET, y compris des scripts côté client.

Les contrôles utilisateur incluent généralement `Control` directive `Control` en plus de sa définition:

```
<%@ Control Language="C#" AutoEventWireup="True" CodeFile="UserControl.ascx.cs" %>
```

Comme les pages ASPX, les contrôles utilisateur sont des balises pouvant être associées à un code derrière un fichier pour effectuer certains événements et certaines tâches. Par conséquent, toutes les balises HTML disponibles sur la page ASPX peuvent être utilisées sur les contrôles utilisateur sauf `<html>`, `<body>` et `<form>` tags

Voici un exemple de balisage simple de contrôle utilisateur:

```
<%-- UserControl.ascx --%>
<%@ Control Language="C#" AutoEventWireup="True" CodeFile="UserControl.ascx.cs" %>
<div>
    <asp:Label ID="Label1" runat="server" />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Click Here" OnClick="Button1_Click" />
</div>
```

Exemple de code-derrière:

```
// UserControl.ascx.cs
public partial class UserControl : System.Web.UI.UserControl
{
    protected void Button1_Click(Object sender, EventArgs e)
```



```
{
    Label1.Text = "Hello World!";
}
```

Avant qu'un contrôle utilisateur inséré dans la page ASPX, la directive `Register` soit déclarée en haut de la page faisant référence au contrôle utilisateur avec son URL source, son nom de tag et son préfixe de tag.

```
<%@ Register Src="UserControl.ascx" TagName="UserControl" TagPrefix="uc" %>
```

Ensuite, vous pouvez placer le contrôle utilisateur dans la page ASPX comme le contrôle serveur intégré ASP.NET:

```
<uc:UserControl ID="UserControl1" runat="server" />
```

Création d'une instance de contrôle utilisateur par programme

Si vous souhaitez instancier une instance de contrôle utilisateur dans le code ASPX derrière la page, vous devez écrire la déclaration de contrôle utilisateur sur l'événement `Page_Load` comme suit:

```
public partial class Default : System.Web.UI.Page
{
    protected void Page_Load(Object sender, EventArgs e)
    {
        Control control1 = LoadControl("UserControl.ascx");
        Page.Controls.Add(control1);
    }
}
```

Notez que le fichier ASCX du contrôle utilisateur doit déjà être créé lors de l'exécution de la méthode `LoadControl`.

Un autre moyen connu pour déclarer des contrôles utilisateur par programme consiste à utiliser `Placeholder` :

```
public partial class Default : System.Web.UI.Page
{
    public Placeholder Placeholder1;
    protected void Page_Load(Object sender, EventArgs e)
    {
        Control control1 = LoadControl("UserControl.ascx");
        Placeholder1.Controls.Add(control1);
    }
}
```

En fonction de vos besoins, `Placeholder` place les contrôles utilisateur sur un conteneur stockant tous les contrôles serveur ajoutés dynamiquement dans la page, où `Page.Controls` insère directement le contrôle utilisateur dans la page, ce qui est préférable pour le rendu des contrôles littéraux HTML.

Ajout de propriétés personnalisées pour le contrôle utilisateur

Tout comme les contrôles serveur ASP.NET standard, les contrôles utilisateur peuvent avoir des propriétés (attributs) sur leur balise de définition. Supposons que vous souhaitiez ajouter un effet de couleur sur le fichier `UserControl.ascx` comme ceci:

```
<uc:UserControl ID="UserControl1" runat="server" Color="blue" />
```

À ce stade, les attributs / propriétés personnalisés des contrôles utilisateur peuvent être définis en déclarant les propriétés dans le code du contrôle utilisateur derrière:

```
private String _color;
public String Color
{
    get
    {
        return _color;
    }
    set
    {
        _color = value;
    }
}
```

En outre, si vous souhaitez définir une valeur par défaut sur une propriété de contrôle utilisateur, affectez la valeur par défaut dans la méthode constructeur du contrôle utilisateur.

```
public UserControl()
{
    _color = "red";
}
```

Ensuite, le balisage du contrôle utilisateur doit être modifié pour ajouter un attribut de couleur comme suit:

```
<%@ Control Language="C#" AutoEventWireup="True" CodeFile="UserControl.ascx.cs" %>
<div>
    <span style="color:<%= Color %>"><asp:Label ID="Label1" runat="server" /></span>
    <br />
    <asp:Button ID="Button1" runat="server" Text="Click Here" OnClick="Button1_Click" />
</div>
```

Lire ASP.NET - Contrôles utilisateur en ligne: <https://riptutorial.com/fr/asp-net/topic/6773/asp-net---controles-utilisateur>

Chapitre 6: ASP.NET - Etat de gestion

Exemples

Afficher l'état

L'exemple suivant illustre le concept de stockage de l'état d'affichage. Gardons un compteur qui est incrémenté chaque fois que la page est publiée en cliquant sur un bouton de la page. Un contrôle d'étiquette indique la valeur dans le compteur.

Le code du fichier de balisage est le suivant:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="statedemo._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

    <head runat="server">
        <title>
            Untitled Page
        </title>
    </head>

    <body>
        <form id="form1" runat="server">

            <div>
                <h3>View State demo</h3>

                Page Counter:

                <asp:Label ID="lblCounter" runat="server" />
                <asp:Button ID="btnIncrement" runat="server" Text="Add Count"
onclick="btnIncrement_Click" />
            </div>

        </form>
    </body>

</html>
```

Le code derrière le fichier de l'exemple est affiché ici:

```
public partial class _Default : System.Web.UI.Page
{
    public int counter
    {
        get
        {
            if (ViewState["pcounter"] != null)
            {
```



```
        return ((int)ViewState["pcounter"]);
    }
    else
    {
        return 0;
    }
}

set
{
    ViewState["pcounter"] = value;
}
}

protected void Page_Load(object sender, EventArgs e)
{
    lblCounter.Text = counter.ToString();
    counter++;
}
}
```

Il produirait le résultat suivant:

Voir la démo d'état

View State demo

Page Counter: 1

Lire ASP.NET - Etat de gestion en ligne: <https://riptutorial.com/fr/asp-net/topic/6296/asp-net---etat-de-gestion>

Chapitre 7: ASP.NET - Validateurs

Syntaxe

- **RequiredFieldValidator Contrôle:** `<asp: RequiredFieldValidator ID = "rfvcandidate" runat = "serveur" ControlToValidate = "ddlcandidate" ErrorMessage = "Veuillez choisir un candidat" InitialValue = "Veuillez choisir un candidat">
</ asp: RequiredFieldValidator>`
- **Contrôle de RangeValidator:**
`<asp: RangeValidator ID = "rvclass" runat = "serveur" ControlToValidate = "txtclass" ErrorMessage = "Entrez votre classe (6 - 12)" MaximumValue = "12" MinimumValue = "6" Type = "Entier">
</ asp: RangeValidator>`
- **Contrôle CompareValidator:** `<asp: CompareValidator ID = "CompareValidator1" runat = "serveur" ErrorMessage = "CompareValidator"> </ asp: CompareValidator>`
- **CustomValidator:**
`<asp: CustomValidator ID = "CustomValidator1" runat = "serveur" ClientValidationFunction = .cvf_func. ErrorMessage = "CustomValidator">
</ asp: CustomValidator>`
- **Résumé de la validation:** `<asp: ValidationSummary ID = "ValidationSummary1" runat = "serveur" DisplayMode = "BulletList" ShowSummary = "true" HeaderText = "Erreurs:" />`

Exemples

Contrôles de validation

Les contrôles de validation ASP.NET valident les données saisies par l'utilisateur pour garantir que les données inutiles, non authentifiées ou contradictoires ne soient pas stockées.

ASP.NET fournit les contrôles de validation suivants:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- RegularExpressionValidator
- CustomValidator
- ValidationSummary

Contrôle RequiredFieldValidator

Le contrôle `RequiredFieldValidator` garantit que le champ requis n'est pas vide. Il est généralement lié à une zone de texte pour forcer la saisie dans la zone de texte.

La syntaxe du contrôle est la suivante:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
    runat="server" ControlToValidate ="ddlcandidate"
    ErrorMessage="Please choose a candidate"
    InitialValue="Please choose a candidate">

</asp:RequiredFieldValidator>
```

Contrôle de `RangeValidator`

Le contrôle `RangeValidator` vérifie que la valeur d'entrée se situe dans une plage prédéterminée.

Il possède trois propriétés spécifiques:

Propriétés	La description
Type	Il définit le type de données. Les valeurs disponibles sont: devise, date,
Valeur minimum	Il spécifie la valeur minimale de la plage.
Valeur maximum	Il spécifie la valeur maximale de la plage.

La syntaxe du contrôle est la suivante:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
    ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
    MinimumValue="6" Type="Integer">

</asp:RangeValidator>
```

`CompareValidator` Control

Le contrôle `CompareValidator` compare une valeur dans un contrôle avec une valeur fixe ou une valeur dans un autre contrôle.

Il possède les propriétés spécifiques suivantes:

Propriétés	La description
Type	Il spécifie le type de données.
ControlToCompare	Il spécifie la valeur du contrôle d'entrée à comparer.
ValueToCompare	Il spécifie la valeur constante à comparer.
ValueToCompare	Il spécifie l'opérateur de comparaison, les valeurs disponibles sont:

Propriétés	La description
	Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual et DataTypeCheck.

La syntaxe de base du contrôle est la suivante:

```
<asp:CompareValidator ID="CompareValidator1" runat="server"
    ErrorMessage="CompareValidator">

</asp:CompareValidator>
```

RegularExpressionValidator

RegularExpressionValidator permet de valider le texte en entrée en le comparant à un modèle d'une expression régulière. L'expression régulière est définie dans la propriété ValidationExpression.

Le tableau suivant résume les constructions de syntaxe couramment utilisées pour les expressions régulières:

Évasion de personnage	La description
\ b	Correspond à un retour arrière.
\ t	Correspond à un onglet.
\ r	Correspond à un retour chariot.
\ v	Correspond à un onglet vertical.
\ F	Correspond à un flux de formulaire.
\ n	Correspond à une nouvelle ligne.
\	Caractère d'échappement.

En dehors de la correspondance avec un seul caractère, une classe de caractères pouvant être mise en correspondance, appelée métacaractères, peut être spécifiée.

Métacaractères	La description
.	Correspond à n'importe quel caractère sauf \ n.
[a B c d]	Correspond à n'importe quel personnage de l'ensemble.
[^ abcd]	Exclut tous les caractères de l'ensemble.
[2-7a-mA-M]	Correspond à tout caractère spécifié dans la plage.

Métacaractères	La description
\w	Correspond à tout caractère alphanumérique et trait de soulignement.
\W	Correspond à n'importe quel caractère non-mot.
\s	Correspond aux espaces tels que les espaces, les tabulations, les nouvelles lignes, etc.
\S	Correspond à n'importe quel caractère autre qu'un espace.
\ré	Correspond à n'importe quel caractère décimal.
\RÉ	Correspond à tout caractère non décimal.

Des quantificateurs peuvent être ajoutés pour spécifier le nombre de fois qu'un caractère peut apparaître.

Quantificateur	La description
*	Zéro ou plus de matches.
+	Un ou plusieurs matches.
?	Zéro ou un match.
{N}	N matches.
{N,}	N ou plus de matches.
{N, M}	Entre N et M correspond.

La syntaxe du contrôle est la suivante:

```
<asp:RegularExpressionValidator ID="string" runat="server" ErrorMessage="string"
    ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>
```

Résumé de validation

Le contrôle ValidationSummary n'effectue aucune validation mais affiche un récapitulatif de toutes les erreurs de la page. Le récapitulatif affiche les valeurs de la propriété ErrorMessage de tous les contrôles de validation ayant échoué à la validation.

Les deux propriétés suivantes, mutuellement incluses, répertorient le message d'erreur:

ShowSummary: affiche les messages d'erreur au format spécifié.

ShowMessageBox: affiche les messages d'erreur dans une fenêtre séparée.

La syntaxe du contrôle est la suivante:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
    DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />
```

Groupes de validation

Les pages complexes comportent différents groupes d'informations fournis dans différents panneaux. Dans ce cas, il peut être nécessaire d'effectuer la validation séparément pour un groupe distinct. Ce type de situation est géré à l'aide de groupes de validation.

Pour créer un groupe de validation, vous devez placer les contrôles d'entrée et les contrôles de validation dans le même groupe logique en définissant leur propriété `ValidationGroup`.

Exemple L'exemple suivant décrit un formulaire à remplir par tous les élèves d'une école, divisée en quatre maisons, pour élire le président de l'école. Ici, nous utilisons les contrôles de validation pour valider l'entrée utilisateur.

C'est la forme en mode conception:

Le code du fichier de contenu est le suivant:

```
<form id="form1" runat="server">

    <table style="width: 66%; ">

        <tr>
            <td class="style1" colspan="3" align="center">
                <asp:Label ID="lblmsg"
                    Text="President Election Form : Choose your president"
                    runat="server" />
            </td>
        </tr>

        <tr>
            <td class="style3">
                Candidate:
            </td>

            <td class="style2">
```



```

        <asp:DropDownList ID="ddlcandidate" runat="server" style="width:239px">
            <asp:ListItem>Please Choose a Candidate</asp:ListItem>
            <asp:ListItem>M H Kabir</asp:ListItem>
            <asp:ListItem>Steve Taylor</asp:ListItem>
            <asp:ListItem>John Abraham</asp:ListItem>
            <asp:ListItem>Venus Williams</asp:ListItem>
        </asp:DropDownList>
    </td>

    <td>
        <asp:RequiredFieldValidator ID="rfvcandidate"
            runat="server" ControlToValidate ="ddlcandidate"
            ErrorMessage="Please choose a candidate"
            InitialValue="Please choose a candidate">
        </asp:RequiredFieldValidator>
    </td>
</tr>

<tr>
    <td class="style3">
        House:
    </td>

    <td class="style2">
        <asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
            <asp:ListItem>Red</asp:ListItem>
            <asp:ListItem>Blue</asp:ListItem>
            <asp:ListItem>Yellow</asp:ListItem>
            <asp:ListItem>Green</asp:ListItem>
        </asp:RadioButtonList>
    </td>

    <td>
        <asp:RequiredFieldValidator ID="rfvhouse" runat="server"
            ControlToValidate="rblhouse" ErrorMessage="Enter your house name" >
        </asp:RequiredFieldValidator>
        <br />
    </td>
</tr>

<tr>
    <td class="style3">
        Class:
    </td>

    <td class="style2">
        <asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
    </td>

    <td>
        <asp:RangeValidator ID="rvclass"
            runat="server" ControlToValidate="txtclass"
            ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
            MinimumValue="6" Type="Integer">
        </asp:RangeValidator>
    </td>
</tr>

<tr>
    <td class="style3">
        Email:

```



```

        </td>

        <td class="style2">
            <asp:TextBox ID="txtemail" runat="server" style="width:250px">
            </asp:TextBox>
        </td>

        <td>
            <asp:RegularExpressionValidator ID="remail" runat="server"
                ControlToValidate="txtemail" ErrorMessage="Enter your email"
                ValidationExpression="\w+([-+.' ]\w+)*@\w+([-.\ ]\w+)*\.\w+([-.\ ]\w+)*">
            </asp:RegularExpressionValidator>
        </td>
    </tr>

    <tr>
        <td class="style3" align="center" colspan="3">
            <asp:Button ID="btnsubmit" runat="server" onclick="btnsubmit_Click"
                style="text-align: center" Text="Submit" style="width:140px" />
        </td>
    </tr>
</table>
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
    DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />
</form>

```

Le code derrière le bouton de soumission:

```

protected void btnsubmit_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        lblmsg.Text = "Thank You";
    }
    else
    {
        lblmsg.Text = "Fill up all the fields";
    }
}

```

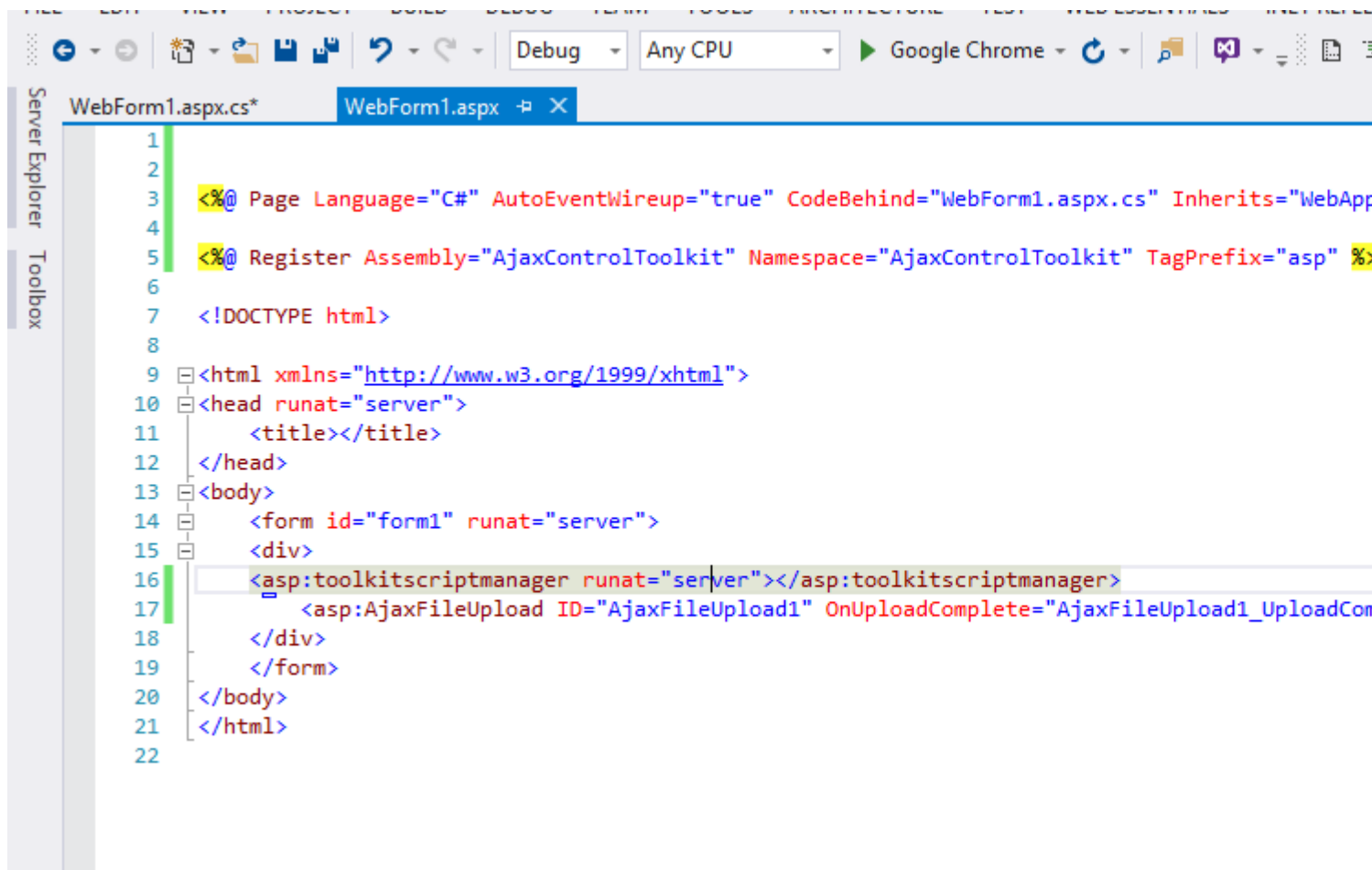
Lire ASP.NET - Validateurs en ligne: <https://riptutorial.com/fr/asp-net/topic/6180/asp-net---validateurs>

Chapitre 8: Contrôles Asp.net Ajax

Exemples

FileUpload Ajax Toolkit Control

1. Ajoutez une référence de `AjaxToolkitControl.dll` dans votre projet.
2. Ensuite, faites glisser et déposez `Toolkit Script Manager` et `AjaxFileUpload Control` depuis la fenêtre Visual Studio Toolbox vers votre page `.aspx` comme ceci:



```
1
2
3 <%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebApp" %>
4
5 <%@ Register Assembly="AjaxControlToolkit" Namespace="AjaxControlToolkit" TagPrefix="asp" %>
6
7 <!DOCTYPE html>
8
9 <html xmlns="http://www.w3.org/1999/xhtml">
10 <head runat="server">
11 <title></title>
12 </head>
13 <body>
14 <form id="form1" runat="server">
15 <div>
16 <asp:toolkitscriptmanager runat="server"></asp:toolkitscriptmanager>
17 <asp:AjaxFileUpload ID="AjaxFileUpload1" OnUploadComplete="AjaxFileUpload1_UploadComplete" />
18 </div>
19 </form>
20 </body>
21 </html>
22
```

3. utilisez ce code sur votre fichier `aspx.cs`


```

using System.Web.UI.WebControls;

namespace WebApplication1
{
    1 reference
    public partial class WebForm1 : System.Web.UI.Page
    {
        0 references
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        0 references
        protected void AjaxFileUpload1_UploadComplete(object sender, AjaxControlToolkit.AjaxFileUploadEventArgs e)
        {
            string fileName = Path.GetFileName(e.FileName);
            AjaxFileUpload1.SaveAs(Server.MapPath("~/Uploads/" + fileName));
        }
    }
}

```

4. Assurez-vous d'avoir créé un dossier nommé **Uploads** dans le répertoire racine de votre projet.

Lire Contrôles Asp.net Ajax en ligne: <https://riptutorial.com/fr/asp-net/topic/7164/contrôles-asp-net-ajax>

Chapitre 9: Cycle de vie de la page

Exemples

Événements du cycle de vie

Voici les événements du cycle de vie de la page:

PreInit - PreInit est le premier événement du cycle de vie d'une page. Il vérifie la propriété `IsPostBack` et détermine si la page est une publication. Il définit les thèmes et les pages maîtres, crée des contrôles dynamiques et obtient et définit les valeurs des propriétés du profil. Cet événement peut être géré en remplaçant la méthode `OnPreInit` ou en créant un gestionnaire `Page_PreInit`.

L' événement **Init** -Init initialise la propriété de contrôle et l'arborescence de contrôle est générée. Cet événement peut être géré en remplaçant la méthode `OnInit` ou en créant un gestionnaire `Page_Init`.

InitComplete - L'événement `InitComplete` permet le suivi de l'état de la vue. Toutes les commandes activent le suivi de l'état d'affichage.

LoadViewState - L'événement `LoadViewState` permet de charger les informations d'état d'affichage dans les contrôles.

LoadPostData - Au cours de cette phase, le contenu de tous les champs d'entrée est défini avec la balise.

PreLoad - PreLoad se produit avant que les données postérieures ne soient chargées dans les contrôles. Cet événement peut être géré en remplaçant la méthode `OnPreLoad` ou en créant un gestionnaire `Page_PreLoad`.

Load - L'événement Load est d'abord déclenché pour la page, puis récursivement pour tous les contrôles enfants. Les contrôles de l'arborescence de contrôle sont créés. Cet événement peut être géré en remplaçant la méthode `OnLoad` ou en créant un gestionnaire `Page_Load`.

LoadComplete - Le processus de chargement est terminé, les gestionnaires d'événements de contrôle sont exécutés et la validation de la page a lieu. Cet événement peut être géré en remplaçant la méthode `OnLoadComplete` ou en créant un gestionnaire `Page_LoadComplete`

PreRender - L'événement PreRender se produit juste avant le rendu de la sortie. En gérant cet événement, les pages et les contrôles peuvent effectuer toutes les mises à jour avant le rendu de la sortie.

PreRenderComplete - Lorsque l'événement PreRender est déclenché de manière récursive pour tous les contrôles enfants, cet événement garantit la fin de la phase de pré-rendu.

SaveStateComplete - L'état du contrôle sur la page est enregistré. Les informations de

personnalisation, d'état de contrôle et d'état d'affichage sont enregistrées. Le balisage HTML est généré. Cette étape peut être gérée en remplaçant la méthode Render ou en créant un gestionnaire Page_Render.

UnLoad - La phase UnLoad est la dernière phase du cycle de vie de la page. Il déclenche l'événement UnLoad pour tous les contrôles de manière récursive et enfin pour la page elle-même. Le nettoyage final est terminé et toutes les ressources et références, telles que les connexions à la base de données, sont libérées. Cet événement peut être géré en remplaçant la méthode OnUnLoad ou en créant un gestionnaire Page_UnLoad.

Exemple de code

```
using System;

namespace myProject
{
    public partial class WebForm1 : System.Web.UI.Page
    {
        public string PageSteps = string.Empty;

        //Raised after the start stage is complete and before the initialization stage begins.
        protected void Page_PreInit(object sender, EventArgs e)
        {
            PageSteps += "1 - Page_PreInit<br>";

            //Access to page Controls not available in this step
            //Label1.Text = "Step 1";
        }

        //Raised after all controls have been initialized and any skin settings have been
        applied.
        //The Init event of individual controls occurs before the Init event of the page.
        protected void Page_Init(object sender, EventArgs e)
        {
            PageSteps += "2 - Page_Init<br>";

            Label1.Text = "Step 2";
        }

        //Raised at the end of the page's initialization stage.
        //Only one operation takes place between the Init and InitComplete events: tracking of
        view state changes is turned on.
        //View state tracking enables controls to persist any values that are programmatically
        added to the ViewState collection.
        //Until view state tracking is turned on, any values added to view state are lost
        across postbacks.
        //Controls typically turn on view state tracking immediately after they raise their
        Init event.
        protected void Page_InitComplete(object sender, EventArgs e)
        {
            PageSteps += "3 - Page_InitComplete<br>";

            Label1.Text = "Step 3";
        }

        //Raised after the page loads view state for itself and all controls, and after it
        processes postback data that is included with the Request instance.
```



```

protected override void OnPreLoad(EventArgs e)
{
    PageSteps += "4 - OnPreLoad<br>";

    Label1.Text = "Step 4";
}

//The Page object calls the OnLoad method on the Page object, and then recursively
does the same for each child control until the page and all controls are loaded.
//The Load event of individual controls occurs after the Load event of the page.
protected void Page_Load(object sender, EventArgs e)
{
    PageSteps += "5 - Page_Load<br>";

    Label1.Text = "Step 5";
}

//Use these events to handle specific control events, such as a Button control's Click
event or a TextBox control's TextChanged event.
protected void btnSubmit_Click(object sender, EventArgs e)
{
    //Step only visible on PostBack
    PageSteps += "6 - btnSubmit_Click<br>";

    Label1.Text = "Step 6";
}

//Raised at the end of the event-handling stage.
protected void Page_LoadComplete(object sender, EventArgs e)
{
    PageSteps += "7 - Page_LoadComplete<br>";

    Label1.Text = "Step 7";
}

//Raised after the Page object has created all controls that are required in order to
render the page, including child controls of composite controls.
//(To do this, the Page object calls EnsureChildControls for each control and for the
page.)
protected override void OnPreRender(EventArgs e)
{
    PageSteps += "8 - OnPreRender<br>";

    Label1.Text = "Step 8";
}

//Raised after each data bound control whose DataSourceID property is set calls its
DataBind method.
protected override void OnPreRenderComplete(EventArgs e)
{
    PageSteps += "9 - OnPreRenderComplete<br>";

    Label1.Text = "Step 9";
}

//Raised after view state and control state have been saved for the page and for all
controls.
//Any changes to the page or controls at this point affect rendering, but the changes
will not be retrieved on the next postback.
protected override void OnSaveStateComplete(EventArgs e)

```



```

    {
        PageSteps += "10 - OnSaveStateComplete<br><hr><br>";

        Label1.Text = "Step 10";
    }

    // Render
    //This is not an event; instead, at this stage of processing, the Page object calls
    this method on each control.
    //All ASP.NET Web server controls have a Render method that writes out the control's
    markup to send to the browser.

    //Raised for each control and then for the page.
    //Controls use this event to do final cleanup for specific controls, such as closing
    control-specific database connections
    protected void Page_UnLoad(object sender, EventArgs e)
    {
        //This last PageSteps addition will not be visible on the page
        PageSteps += "11 - Page_UnLoad<br>";

        //Access to page Controls not available in this step
        //Label1.Text = "Step 11";
    }
}

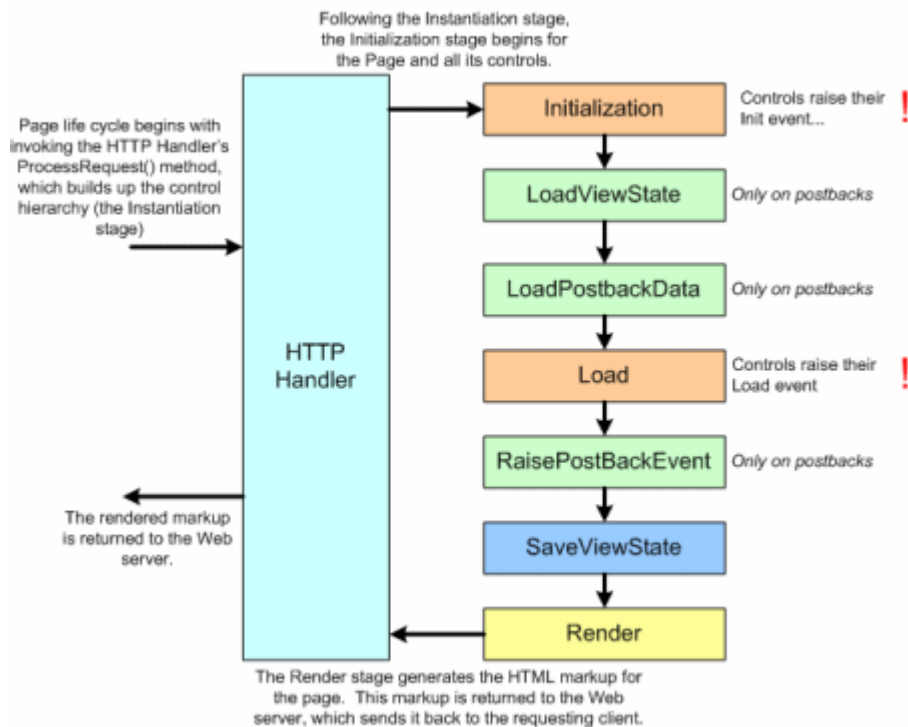
```

Ajoutez le code suivant à la page .aspx pour visualiser les étapes du cycle de vie.

```

<b>Page Life Cycle Visualization:</b>
<br />
<%= PageSteps %>

```

Plus d'information

- <https://msdn.microsoft.com/en-us/library/ms178472.aspx>
- https://www.tutorialspoint.com/asp.net/asp.net_life_cycle.htm
- <http://www.c-sharpcorner.com/UploadFile/8911c4/page-life-cycle-with-examples-in-Asp-Net/>
- <https://www.codeproject.com/Articles/667308/ASP-NET-Page-Life-Cycle-Events>

Lire Cycle de vie de la page en ligne: <https://riptutorial.com/fr/asp-net/topic/4948/cycle-de-vie-de-la-page>

Chapitre 10: Délégation d'événement

Syntaxe

1. `public delegate void ActionClick();`

```
public event ActionClick OnResetClick;
```

Remarques

Je n'ai pas trouvé d'inconvénient dans cette approche mais il y a quelques petites choses qui rendent cela un peu problématique.

1. Vous devez ajouter un gestionnaire d'événement pour chaque événement. Si vous n'ajoutez pas les gestionnaires d'événements dans l'événement OnInit de la page, vous risquez de rencontrer des problèmes qui, lors de la publication de la page, vous feront perdre l'affectation d'événement (car ASP.NET est sans état, ce qui n'est pas le cas avec les contrôles Windows) .
2. Dans cette approche, vous devez respecter les événements du cycle de vie des pages. Parfois, lorsque vous travaillez sur le concepteur, il se peut que le gestionnaire d'événements se perde sans votre avis.
3. Même si vous n'avez pas ajouté le gestionnaire d'événements, vous n'obtiendrez aucune erreur ou aucun avertissement. Si vous avez plusieurs pages pour effectuer la même action, rien ne garantit que tous les noms de méthode seront identiques. Le développeur peut choisir ses propres noms de méthode, ce qui réduit la maintenabilité du code.

Exemples

Délégation d'événement du contrôle utilisateur à aspx

Normalement, nous optons pour cette approche si nous voulons une encapsulation complète et ne souhaitons pas rendre nos méthodes publiques.

Ascx

```
<div style="width: 100%; ">
  <asp:Button ID="btnAdd" runat="server"
    Text="Add" OnClick="btnAdd_Click"></asp:button>
  <asp:button id="btnEdit" runat="server"
    text="Edit" onclick="btnEdit_Click"> </asp:button>
  <asp:button id="btnDelete" runat="server"
    text="Delete" onclick="btnDelete_Click"> </asp:Button>
  <asp:button id="btnReset" runat="server"
    text="Reset" onclick="btnReset_Click"></asp:button>
</div>
```


Ascx.cs

```
public delegate void ActionClick();

public partial class EventDelegation : System.Web.UI.UserControl
{
    public event ActionClick OnAddClick;
    public event ActionClick OnDeleteClick;
    public event ActionClick OnEditClick;
    public event ActionClick OnResetClick;
    protected void btnAdd_Click(object sender, EventArgs e)
    {
        if (OnAddClick != null)
        {
            OnAddClick();
        }
    }

    protected void btnEdit_Click(object sender, EventArgs e)
    {
        if (OnEditClick != null)
        {
            OnEditClick();
        }
    }

    protected void btnDelete_Click(object sender, EventArgs e)
    {
        if (OnDeleteClick != null)
        {
            OnDeleteClick();
        }
    }

    protected void btnReset_Click(object sender, EventArgs e)
    {
        if (OnResetClick != null)
        {
            OnResetClick();
        }
    }
}
```

Le contrôle utilisateur spécifie certains événements publics tels que `OnAddClick`, `OnEditClick`, etc., qui déclarent un délégué. Toute personne souhaitant utiliser ces événements doit ajouter le gestionnaire d'événements à exécuter lorsque l'événement de clic sur le bouton correspondant se produit.

Aspx Design

```
<%@ Register src="Controls/EventDelegation.ascx"
    tagname="EventDelegation" tagprefix="uc1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
```



```

        <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <uc1:Direct ID="Direct1" runat="server" />
        </div>
    </form>
</body>
</html>

```

Aspx.cs

```

public partial class EventDelegation : System.Web.UI.Page
{
    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);
        ActionControl.OnAddClick += ActionControl_OnAddClick;
        ActionControl.OnDeleteClick += ActionControl_OnDeleteClick;
        ActionControl.OnEditClick += ActionControl_OnEditClick;
        ActionControl.OnResetClick += ActionControl_OnResetClick;
    }

    private void ActionControl_OnResetClick()
    {
        Response.Write("Reset done.");
    }

    private void ActionControl_OnEditClick()
    {
        Response.Write("Updated.");
    }

    private void ActionControl_OnDeleteClick()
    {
        Response.Write("Deleted.");
    }

    private void ActionControl_OnAddClick()
    {
        Response.Write("Added.");
    }
}

```

Lire Délégation d'événement en ligne: <https://riptutorial.com/fr/asp-net/topic/6927/delegation-d-evenement>

Chapitre 11: Directives

Exemples

La directive d'application

La directive Application définit des attributs spécifiques à l'application. Il est fourni en haut du fichier global.aspx. La syntaxe de base de la directive Application est la suivante:

```
<%@ Application Language="C#" %>
```

Les attributs de la directive d'application sont les suivants:

Les attributs	La description
Héritiers	Le nom de la classe dont hériter.
La description	La description textuelle de l'application. Les analyseurs et les compilateurs ignorent cela.
La langue	La langue utilisée dans les blocs de code.

La directive de contrôle

La directive de contrôle est utilisée avec les contrôles utilisateur et apparaît dans les fichiers de contrôle utilisateur (.ascx).

La syntaxe de base de la directive de contrôle est la suivante:

```
<%@ Control Language="C#" EnableViewState="false" %>
```

Les attributs de la directive de contrôle sont les suivants:

Les attributs	La description
AutoEventWireup	La valeur booléenne qui active ou désactive l'association automatique des événements aux gestionnaires.
Nom du cours	Le nom du fichier pour le contrôle.
Déboguer	La valeur booléenne qui active ou désactive la compilation avec des symboles de débogage.
La description	La description textuelle de la page de contrôle, ignorée par le compilateur.

Les attributs	La description
EnableViewState	La valeur booléenne qui indique si l'état d'affichage est maintenu à travers les demandes de page.
Explicite	Pour le langage VB, indique au compilateur d'utiliser le mode explicite de l'option.
Héritiers	La classe dont la page de contrôle hérite.
La langue	La langue pour le code et le script.
Src	Le nom de fichier pour la classe code-behind.
Strict	Pour le langage VB, indique au compilateur d'utiliser l'option mode strict.

La directive sur les outils

La directive `Implement` indique que la page Web, la page maître ou la page de contrôle de l'utilisateur doivent implémenter l'interface de framework .Net spécifiée.

La directive de base sur la syntaxe des implémentations est la suivante:

```
<%@ Implements Interface="interface_name" %>
```

La directive maîtresse

La directive principale spécifie un fichier de page comme étant la page du matériel.

La syntaxe de base de l'exemple de directive `MasterPage` est la suivante:

```
<%@ MasterPage Language="C#" AutoEventWireup="true" CodeFile="SiteMater.master.cs"
Inherits="SiteMaster" %>
```

La directive sur les importations

La directive `Import` importe un espace de noms dans une page Web, page de contrôle utilisateur de l'application. Si la directive `Import` est spécifiée dans le fichier global.asax, elle est appliquée à l'ensemble de l'application. S'il se trouve dans une page de page de contrôle utilisateur, il est alors appliqué à cette page ou à ce contrôle.

La syntaxe de base pour la directive d'importation est la suivante:

```
<%@ namespace="System.Drawing" %>
```

La directive `MasterType`

La directive `MasterType` attribue un nom de classe à la propriété `Master` d'une page pour la rendre

fortement typée.

La syntaxe de base de la directive MasterType est la suivante:

```
<%@ MasterType attribute="value" [attribute="value" ...] %>
```

La directive page

La directive Page définit les attributs spécifiques au fichier de page pour l'analyseur de page et le compilateur.

La syntaxe de base de la directive page est la suivante:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" Trace="true" %>
```

Les attributs de la directive Page sont les suivants:

Les attributs	La description
AutoEventWireup	Valeur booléenne qui active ou désactive les événements de page liés automatiquement aux méthodes. Par exemple, Page_Load.
Tampon	Valeur booléenne qui active ou désactive la mise en mémoire tampon des réponses HTTP.
Nom du cours	Le nom de la classe pour la page.
ClientTarget	Le navigateur pour lequel les contrôles du serveur doivent afficher le contenu.
CodeFile	Le nom du code derrière le fichier.
Déboguer	Valeur booléenne qui active ou désactive la compilation avec des symboles de débogage.
La description	La description textuelle de la page, ignorée par l'analyseur.
EnableSessionState	Il active, désactive ou rend l'état de session en lecture seule.
EnableViewState	Valeur booléenne qui active ou désactive l'état d'affichage sur les requêtes de page.
ErrorPage	URL de redirection si une exception de page non gérée se produit.
Héritiers	Le nom du code derrière ou autre classe.
La langue	Le langage de programmation pour le code.
Src	Le nom de fichier du code derrière la classe.

Les attributs	La description
Trace	Il active ou désactive le traçage.
TraceMode	Il indique comment les messages de trace sont affichés et triés par heure ou par catégorie.
Transaction	Il indique si les transactions sont prises en charge.
ValidateRequest	Valeur booléenne indiquant si toutes les données d'entrée sont validées par rapport à une liste de valeurs codées en dur.

La directive OutputCache

La directive OutputCache contrôle les règles de mise en cache de sortie d'une page Web ou d'un contrôle utilisateur.

La syntaxe de base de la directive OutputCache est la suivante:

```
<%@ OutputCache Duration="15" VaryByParam="None" %>
```

Lire Directives en ligne: <https://riptutorial.com/fr/asp-net/topic/2255/directives>

Chapitre 12: Etat de session

Syntaxe

- `Session ["Session_Key"] = Obj_Value;`

Remarques

HTTP est sans état. L'état de session ASP.NET est une structure qui facilite la gestion de l'état entre les requêtes de page HTTP.

Session diffère des variables de niveau de classe par sa capacité à rester disponible dans les post-back et les différentes pages. Par exemple, une variable de session créée dans `Page1.aspx` sera disponible si l'utilisateur est redirigé vers `Page2.aspx` dans la même application.

De plus, contrairement aux variables statiques déclarées au niveau de la page, les variables de session sont indépendantes pour différents utilisateurs. Cela signifie que la modification de la valeur de la variable de session d'un utilisateur n'affectera pas la valeur de la même variable pour les autres utilisateurs.

Bien que `ViewState` puisse être utilisé pour stocker les données de l'utilisateur temporairement, il ne permet pas d'enregistrer des données sur plusieurs pages. En outre, le `viewstate` fait partie de la page et est envoyé au client. En conséquence, toute information critique liée à l'utilisateur ne peut pas être enregistrée dans `ViewState`, et c'est là que les variables de session deviennent utiles.

Exemples

Utilisation de l'objet Session pour stocker des valeurs

L'objet `System.Web.SessionState.HttpSessionState` permet de conserver les valeurs entre les requêtes HTTP. Dans l'exemple ci-dessous, la préférence d'un utilisateur pour les avertissements est enregistrée dans la session. Plus tard, lors de l'envoi d'une autre requête à l'utilisateur, l'application peut lire cette préférence depuis la session et masquer les avertissements.

```
public partial class Default : System.Web.UI.Page
{
    public void LoadPreferences(object sender, EventArgs args)
    {
        // ...
        // ... A DB operation that loads the user's preferences.
        // ...

        // Store a value with the key showWarnings
        HttpContext.Current.Session["showWarnings"] = false;
    }
}
```



```

public void button2Clicked(object sender, EventArgs args)
{
    // While processing another request, access this value.
    bool showWarnings = (bool)HttpContext.Current.Session["showWarnings"];
    lblWarnings.Visible = false;
}
}

```

Notez que les variables de session ne sont pas communes à tous les utilisateurs (tout comme les cookies) et qu'elles sont conservées sur plusieurs messages postés.

La session fonctionne en définissant un cookie contenant un identifiant pour la session des utilisateurs. Par défaut, cet identifiant est stocké dans la mémoire du serveur Web, avec les valeurs stockées sur celui-ci.

Voici une capture d'écran du cookie défini dans le navigateur de l'utilisateur pour suivre la session:

Name	Value
ASP.NET_SessionId	3235CC720E020D5F045

Utilisation d'un magasin de sessions SQL

Si vous constatez que plusieurs serveurs doivent partager l'état de la session, leur stockage dans la mémoire de processus ASP.NET ne fonctionnera pas. Par exemple, vous pouvez déployer dans un environnement de batterie de serveurs Web avec un équilibreur de charge qui distribue les demandes à tour de rôle. Dans cet environnement, les requêtes d'un seul utilisateur peuvent être servies par plusieurs serveurs.

Dans le fichier web.config, vous pouvez configurer un magasin de sessions SQL Server.

```

<configuration>
  <system.web>
    <sessionState
      mode="SQLServer"
      sqlConnectionString="Data Source=localhost;Integrated Security=SSPI"
      cookieless="true"
      timeout="30" />
    </system.web>
  </configuration>

```

Pour créer le schéma SQL, utilisez l'outil aspnet_regsql. [SampleSqlServerName] est le nom d'hôte du serveur SQL. -ssadd indique à l'outil de créer la base de données d'état de session. -sstype p indique à l'outil de créer une nouvelle base de données avec le nom par défaut ASPState.

```
aspnet_regsql.exe -S [SampleSqlServerName] -U [Username] -P [Password] -ssadd -sstype p
```

Utilisation d'un magasin de sessions Amazon DynamoDB

Si vous ne souhaitez pas utiliser SQL Server, vous pouvez utiliser la base de données nosql Dynamo DB hébergée d'Amazon en tant que magasin de sessions.

Vous aurez besoin du kit SDK AWS. Pour l'installer à partir de la console du gestionnaire de packages Visual Studio nuget, utilisez la commande suivante

```
Install-Package AWSSDK
```

Vous pouvez ensuite configurer votre fournisseur sessionState pour utiliser un fournisseur personnalisé. Vous devez spécifier la région et les informations d'identification, soit un profil ou une combinaison d'accès IAM et de clé secrète. Par défaut, cela créera une table nommée ASP.NET_SessionState.

```
<configuration>
  <system.web>
    <sessionState
      timeout="20"
      mode="Custom"
      customProvider="DynamoDBSessionStoreProvider">
      <providers>
        <add name="DynamoDBSessionStoreProvider"
          type="Amazon.SessionProvider.DynamoDBSessionStateStore"
          AWSProfileName="[PROFILE]"
          Region="[REGION]"
          CreateIfNotExist="true"
          />
      </providers>
    </sessionState>
  </system.web>
</configuration>
```

Lire Etat de session en ligne: <https://riptutorial.com/fr/asp-net/topic/3864/etat-de-session>

Chapitre 13: Expressions

Exemples

Valeur de App.Config

```
<asp:Literal runat="server" text="<%$ AppSettings:MyAppSettingName %>" />
```

Expression évaluée

```
<div>
  The time is now <%= DateTime.Now.ToString() %>
</div>
```

Bloc de code dans le balisage ASP

```
<div>
  <form id="form1" runat="server">
    <%
      for (int i = 1; i <= 10; j++)
      {
        Response.Write(i) + " ";
      }
    %>
  </form>
</div>
```

Lire Expressions en ligne: <https://riptutorial.com/fr/asp-net/topic/6326/expressions>

Chapitre 14: Gestion de session

Exemples

Avantage et désavantage de l'état de session, types de session

The advantages of using Session State are

- 1) Better security
- 2) Reduced bandwidth

The disadvantages of using Session state are

- 1) More resource consumption of server.
- 2) Extra code/care if a Web farm is used (we will discuss this shortly)

****Session State Modes****

1) InProc mode, which stores session state in memory on the Web server. This is the default.

2) StateServer mode, which stores session state in a separate process called the ASP.NET state service. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm.

3) SQLServer mode stores session state in a SQL Server database. This ensures that session state is preserved if the Web application is restarted and also makes session state available to multiple Web servers in a Web farm.

4) Custom mode, which enables you to specify a custom storage provider.

Off mode, which disables session state.

Lire Gestion de session en ligne: <https://riptutorial.com/fr/asp-net/topic/4180/gestion-de-session>

Chapitre 15: Gestion des événements

Syntaxe

- void privé EventName (expéditeur d'objet, EventArgs e);

Paramètres

Paramètre	Détails
expéditeur d'objet	l'expéditeur fait référence à l'objet qui a appelé l'événement qui a déclenché le gestionnaire d'événement. Ceci est utile si vous avez beaucoup d'objets utilisant le même gestionnaire d'événements.
EventArgs e	EventArgs est une classe de base factice. En soi, il est plus ou moins inutile, mais si vous en dérivez, vous pouvez ajouter les données nécessaires à vos gestionnaires d'événements.

Exemples

Événements d'application et de session

Les événements d'application les plus importants sont:

Application_Start - Il est déclenché lors du démarrage de l'application / du site Web.

Application_End - Il est déclenché lorsque l'application / le site Web est arrêté.

De même, les événements de session les plus utilisés sont les suivants:

Session_Start - Il est **déclenché** lorsqu'un utilisateur demande pour la première fois une page à partir de l'application.

Session_End - Il est **déclenché** à la fin de la session.

Événements de page et de contrôle

Les événements communs de page et de contrôle sont les suivants:

DataBinding - Il est **déclenché** lorsqu'un contrôle se lie à une source de données.

Disposé - Il est déclenché lorsque la page ou le contrôle est libéré.

Error - C'est un événement de page qui se produit lorsqu'une exception non gérée est générée.

Init - Il est déclenché lorsque la page ou le contrôle est initialisé.

Load - Il est déclenché lorsque la page ou un contrôle est chargé.

PreRender - Il est **déclenché** lorsque la page ou le contrôle doit être rendu.

Décharger - Il est déclenché lorsque la page ou le contrôle est déchargé de la mémoire.

Événements par défaut

L'événement par défaut de l'objet Page est l'événement Load. De même, chaque contrôle a un événement par défaut. Par exemple, l'événement par défaut pour le contrôle du bouton est l'événement Click.

Le gestionnaire d'événements par défaut peut être créé dans Visual Studio, simplement en double-cliquant sur le contrôle en mode Création. Le tableau suivant présente certains des événements par défaut pour les contrôles communs:

Contrôle	Événement par défaut
AdRotator	AdCreated
Liste à puces	Cliquez sur
Bouton	Cliquez sur
Calandre	SelectionChanged
CheckBox	CheckedChanged
CheckBoxList	SelectedIndexChanged
Grille de données	SelectedIndexChanged
DataList	SelectedIndexChanged
La liste déroulante	SelectedIndexChanged
HyperLink	Cliquez sur
ImageButton	Cliquez sur
ImageMap	Cliquez sur
LinkButton	Cliquez sur
ListBox	SelectedIndexChanged
Menu MenuItem	Cliquez sur
Bouton radio	CheckedChanged
RadioButtonList	SelectedIndexChanged

Exemple Cet exemple comprend une page simple avec un contrôle d'étiquette et un contrôle de bouton. Comme les événements de page tels que Page_Load, Page_Init, Page_PreRender etc. ont lieu, il envoie un message, qui est affiché par le contrôle d'étiquette. Lorsque l'utilisateur clique sur le bouton, l'événement Button_Click est déclenché et envoie également un message à afficher sur l'étiquette.

Créez un nouveau site Web et faites glisser un contrôle d'étiquette et un contrôle de bouton à partir de la boîte à outils de contrôle. A l'aide de la fenêtre des propriétés, définissez les ID des contrôles en tant que .lblmessage. et .btnclick. respectivement. Définissez la propriété Text du contrôle Button en tant que "Click".

Le fichier de balisage (.aspx):

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
    Inherits="eventdemo._Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >

    <head runat="server">
        <title>Untitled Page</title>
    </head>

    <body>
        <form id="form1" runat="server">
            <div>
                <asp:Label ID="lblmessage" runat="server" >

                </asp:Label>

                <br />
                <br />
                <br />

                <asp:Button ID="btnclick" runat="server" Text="Click" onclick="btnclick_Click" />
            </div>
        </form>
    </body>

</html>
```

Double-cliquez sur la vue de conception pour passer au code derrière le fichier. L'événement Page_Load est automatiquement créé sans code. Notez les lignes de code explicatives suivantes:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;

using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
```



```

using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

using System.Xml.Linq;

namespace eventdemo {

    public partial class _Default : System.Web.UI.Page {

        protected void Page_Load(object sender, EventArgs e) {
            lblmessage.Text += "Page load event handled. <br />";

            if (Page.IsPostBack) {
                lblmessage.Text += "Page post back event handled.<br/>";
            }
        }

        protected void Page_Init(object sender, EventArgs e) {
            lblmessage.Text += "Page initialization event handled.<br/>";
        }

        protected void Page_PreRender(object sender, EventArgs e) {
            lblmessage.Text += "Page prerender event handled. <br/>";
        }

        protected void btnclick_Click(object sender, EventArgs e) {
            lblmessage.Text += "Button click event handled. <br/>";
        }
    }
}

```

Exécutez la page. L'étiquette indique le chargement de la page, l'initialisation de la page et les événements de pré-rendu de la page. Cliquez sur le bouton pour voir l'effet:



Lire Gestion des événements en ligne: <https://riptutorial.com/fr/asp-net/topic/2347/gestion-des-evenements>

Chapitre 16: GridView

Exemples

Liaison de données

Il existe deux manières de lier un GridView. Vous pouvez soit le faire manuellement en définissant la propriété `DataSource` et en appelant `DataBind()`, ou vous pouvez utiliser un `DataSourceControl` tel que `SqlDataSource`.

Reliure manuelle

Créez votre GridView:

```
<asp:GridView ID="gvColors" runat="server"></asp:GridView>
```

Commencez par créer ou récupérer les données source pour GridView. Ensuite, affectez les données à la propriété `DataSource` du GridView. Enfin, appelez `DataBind()`.

```
List<string> colors = new List<string>();
colors.Add("Red");
colors.Add("Green");
colors.Add("Blue");

gvColors.DataSource = colors;
gvColors.DataBind();
```

DataSourceControl

Créez votre DataSourceControl:

```
<asp:SqlDataSource ID="sdsColors"
    runat="server"
    ConnectionString="<%= MyConnectionString %>"
    SelectCommand="SELECT Color_Name FROM Colors">
</asp:SqlDataSource>
```

Créez votre GridView et définissez la propriété `DataSourceID`:

```
<asp:GridView ID="gvColors"
    runat="server"
    DataSourceID="sdsColors">
</asp:GridView>
```

Colonnes

Il existe sept types de colonne différents pouvant être utilisés dans GridView.


```
<asp:GridView ID="GridView1" runat="server">
    <Columns>
        ...
    </Columns>
</asp:GridView>
```

BoundField:

```
<asp:BoundField DataField="EmployeeID" HeaderText="Employee ID" />
```

ButtonField:

```
<asp:ButtonField ButtonType="Button" HeaderText="Select Employee" Text="Select"/>
```

CheckBoxField:

```
<asp:CheckBoxField DataField="IsActive" HeaderText="Is Active" />
```

CommandField:

```
<asp:CommandField ShowDeleteButton="true"
    ShowEditButton="true"
    ShowInsertButton="true"
    ShowSelectButton="true" />
```

HyperLinkField:

```
<asp:HyperLinkField HeaderText="Employee Profile"
    DataNavigateUrlFields="EmployeeID"
    DataNavigateUrlFormatString="EmployeeProfile.aspx?EmployeeID={0}" />
```

ImageField:

```
<asp:ImageField HeaderText="Photo"
    DataImageUrlField="EmployeeID"
    DataImageUrlFormatString="/images/{0}" />
```

TemplateField:

```
<asp:TemplateField>
    <HeaderTemplate>
        Name
    </HeaderTemplate>
    <ItemTemplate>
        <asp:Label ID="lblEmployeeName"
            runat="server"
            Text='<&# Eval("EmployeeName") %>'></asp:Label>
    </ItemTemplate>
</asp:TemplateField>
```

GridView fortement typé

En commençant par Asp.net 4.5, les contrôles Web peuvent tirer parti de la liaison fortement typée pour obtenir une prise en charge d'IntelliSense et des erreurs de compilation.

Créez une classe qui contient votre modèle:

```
public class Album
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Artist { get; set; }
}
```

Définissez le contrôle GridView sur votre page:

```
<asp:GridView ID="Grid" runat="server" AutoGenerateColumns="false"
ItemType="YourNamespace.Album">
    <Columns>
        <asp:TemplateField HeaderText="Id">
            <ItemTemplate>
                <asp:Label ID="lblName" runat="server" Text="<%# Item.Id %>"></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Name">
            <ItemTemplate>
                <asp:Label ID="lblName" runat="server" Text="<%# Item.Name %>"></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="Artist">
            <ItemTemplate>
                <asp:Label ID="lblCity" runat="server" Text="<%# Item.Artist %>"></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

Chargez les données et liez-les:

```
var albumList = new List<Album>
{
    new Album {Id = 1, Artist = "Icing (a Cake cover band)", Name = "Toppings Vol. 1"},
    new Album {Id = 2, Artist = "Fleetwood PC", Name = "Best of Windows"},
    new Album {Id = 3, Artist = "this.Bandnames", Name = "TH_ (Pronounced \"Thunderscore\")"},
};

Grid.DataSource = albumList;
Grid.DataBind();
```

Événement de commande de manipulation

GridViews permet d'envoyer des commandes à partir d'une ligne GridView. Ceci est utile pour transmettre des informations spécifiques à une ligne dans un gestionnaire d'événement en tant qu'arguments de commande.

Pour vous abonner à un événement de commande:


```
<asp:GridView ID="GridView1" ... OnRowCommand="GridView1_RowCommand">
```

Les boutons sont le moyen le plus courant de déclencher des commandes. Ils prennent également en charge un moyen de spécifier des arguments de commande. Dans cet exemple, l'argument est l' `ID` de l'élément représenté par la ligne.

```
<TemplateField>
  <ItemTemplate>
    <asp:LinkButton ID="LinkButton1" runat="server"
      CommandName="SampleCmd"
      CommandArgument='<%# Eval("ID") %>'>
    </asp:LinkButton>
  </ItemTemplate>
</TemplateField>
```

Vous pouvez également utiliser un `CommandField` colonne `CommandField` qui fournit les contrôles de commande les plus courants.

Gestion de l'événement en code derrière:

```
protected void GridView1_RowCommand(object source, GridViewCommandEventArgs e)
{
    if (e.CommandName == "SampleCmd")
    {
        var id = e.CommandArgument;
    }
}
```

Notez que le `CommandName` utilisé dans cet exemple est arbitraire et est un choix du développeur. Il existe toutefois un ensemble de noms prédéfinis que `GridView` reconnaît lui-même. Les événements correspondants sont déclenchés lorsque ces commandes sont déclenchées.

Nom de la commande	Événements soulevés
Annuler	RowCancelingEdit
Effacer	RowDeleting, RowDeleted
modifier	RowEditing
Page	PageIndexChanging, PageIndexChanged
Sélectionner	SelectedIndexChanging, SelectedIndexChanged
Trier	Tri, tri
Mettre à jour	RowUpdating, RowUpdated

Pagination

ObjectDataSource

Si vous utilisez un objet ObjectDataSource, presque tout est déjà géré pour vous, il vous suffit de dire à GridView d' `AllowPaging` et de lui donner une `PageSize` .

```
<asp:GridView ID="gvColors"
    runat="server"
    DataSourceID="sdsColors"
    AllowPaging="True"
    PageSize="5">
</asp:GridView>

<asp:SqlDataSource ID="sdsColors"
    runat="server"
    ConnectionString="<%= MyConnectionString %>"
    SelectCommand="SELECT Color_ID, Color_Name FROM Colors">
</asp:SqlDataSource>
```

Color_ID	Color_Name	Color_ID	Color_Name	Color_ID	Color_Name
1	Red	6	Orange	11	Pink
2	Blue	7	Black	12	Turquoise
3	Green	8	White	13	Maroon
4	Yellow	9	Gray		
5	Purple	10	Brown		
123		123		123	

Reliure manuelle

Si la liaison est manuelle, vous devez gérer l'événement `PageIndexChanging` . `PageIndex` simplement le `DataSource` et le `PageIndex` et `PageIndex` le `GridView` .

```
<asp:GridView ID="gvColors"
    runat="server"
    AllowPaging="True"
    PageSize="5"
    OnPageIndexChanging="gvColors_PageIndexChanging">
</asp:GridView>
```

C

```
protected void gvColors_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    gvColors.DataSource = // Method to retrieve DataSource
    gvColors.PageIndex = e.NewPageIndex;
    gvColors.DataBind();
}
```

VB.NET

```
Protected Sub gvColors_PageIndexChanging(sender As Object, e As GridViewPageEventArgs)
{
```



```
gvColors.DataSource = // Method to retrieve DataSource
gvColors.PageIndex = e.NewPageIndex
gvColors.DataBind()
}
```

Cliquez sur Mettre à jour la grille en ligne

Les Gridviews sont plus utiles si nous pouvons mettre à jour la vue selon nos besoins. Considérez une vue avec une fonctionnalité de verrouillage / déverrouillage dans chaque ligne. Cela peut être fait comme:

Ajouter un panneau de mise à jour:

```
<asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional"> </asp:UpdatePanel>
```

Ajoutez un ContentTemplate et un Trigger dans votre UpdatePanel:

```
<asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional">
  <ContentTemplate>
  </ContentTemplate>

  <Triggers>
  </Triggers>
</asp:UpdatePanel>
```

Ajoutez votre GridView à l'intérieur de ContentTemplate:

```
<ContentTemplate>
<asp:GridView ID="GridView1" runat="server">
  <Columns>
    <asp:TemplateField>
      <ItemTemplate>
        <asp:ImageButton ID="imgDownload" runat="server" OnClientClick="return
confirm('Are you sure want to Lock/Unlock ?');"
          CommandName="togglelock"
          CommandArgument='<%=Container.DataItemIndex%>' />

      </ItemTemplate>
    </asp:TemplateField>
  </Columns>

</ContentTemplate>
```

Ici, nous donnons à notre GridView1 une colonne constante, pour le bouton de verrouillage. Attention, la base de données n'a pas encore eu lieu.

Time for DataBind: (sur PageLoad)

```
using (SqlConnection con= new SqlConnection(connectionString))
{
    SqlCommand sqlCommand = new SqlCommand(" ... ", con);
    SqlDataReader reader = sqlCommand.ExecuteReader();
    GridView1.DataSource = reader;
}
```



```

        GridView1.DataBind();
    }

```

Verrouiller / Déverrouiller l'image sera différent selon la valeur d'une certaine colonne de votre GridView. Considérons un cas où votre table contient un attribut / colonne intitulé "Statut de verrouillage". Maintenant, vous souhaitez (1) masquer cette colonne juste après DataBind et juste avant le rendu de la page et (2) Attribuer des images différentes à chaque ligne en fonction de cette valeur de colonne masquée, c.-à-d. jpg ", si le statut est 1, associez-le " unlock.jpg ". Pour ce faire, nous utiliserons l'option `OnRowDataBound` de GridView, il se mêle à votre GridView, juste avant le rendu de chaque ligne dans la page HTML.

```

<ContentTemplate>
<asp:GridView ID="GridView1" runat="server" OnRowDataBound="GridView1_RowDataBound"> ...

```

Dans un fichier cs

```

protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        e.Row.Cells[8].Visible = false; //hiding the desired column which is column number
6 in this case
        GridView1.HeaderRow.Cells[8].Visible = false; //hiding its header
        ImageButton imgDownload = (ImageButton)e.Row.FindControl("imgDownload");
        string lstate = ((CheckBox)e.Row.Cells[8].Controls[0]).Checked.ToString();
        if (lstate == "True")
        { imgDownload.ImageUrl = "images/lock.png"; }
        else
        {
            imgDownload.ImageUrl = "images/unlock.png";
        }
    }
}

```

Maintenant, le GridView sera rendu comme nous le souhaitons, maintenant, laissez-nous implémenter les événements de clic de bouton sur le bouton Verrouiller / Déverrouiller l'image. Comprenez que pour effectuer une opération spécifique sur une ligne spécifique, une commande doit être donnée à cette ligne et GridView nous fournit les mêmes fonctionnalités que `OnRowCommand`.

```

<ContentTemplate>
<asp:GridView ID="GridView1" runat="server" OnRowDataBound="GridView1_RowDataBound"
OnRowCommand="GridView1_RowCommand">
...
</ContentTemplate>

```

Cela va créer une fonction dans le fichier cs qui prend un `object sender` et `GridViewCommandEventArgs e` Avec `e.CommandArgument` nous pouvons obtenir l'indice de la ligne qui a donné la commande point à noter ici est que, une ligne peut avoir plusieurs boutons et le cs Le code doit savoir quel bouton de cette ligne a donné la commande. Nous allons donc utiliser `CommandName`


```
<asp:ImageButton ID="imgDownload" runat="server" OnClientClick="return confirm('Are you sure  
want to Lock/Unlock ?');"  
CommandName="togglelock"  
CommandArgument='<%#Container.DataItemIndex%>' />
```

Maintenant, dans le backend on peut distinguer les commandes des différentes lignes et des différents boutons.

```
protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs e)  
{  
    if (e.CommandName == "togglelock")  
    {  
        using (SqlConnection con= new SqlConnection(connectionString))  
        {  
            int index = Convert.ToInt32(e.CommandArgument);  
            SqlCommand sqlCommand = new SqlCommand(" ... ", con);  
            SqlDataReader reader = sqlCommand.ExecuteReader();  
            GridView1.DataSource = reader;  
            GridView1.DataBind();  
        }  
    }  
}
```

Ajoutez `<asp:PostBackTrigger ControlID="GridView1"/>` au Trigger et il mettra à jour GridView une fois le DataBind terminé.

Utilisez `HorizontalAlign="Center"` pour placer le GridView au centre de la page.

Lire GridView en ligne: <https://riptutorial.com/fr/asp-net/topic/1680/gridview>

Chapitre 17: httpHandlers

Exemples

Utiliser un httpHandler (.ashx) pour télécharger un fichier à partir d'un emplacement spécifique

Créez un nouveau httpHandler dans votre projet ASP.NET. Appliquez le code suivant (VB) au fichier du gestionnaire:

```
Public Class AttachmentDownload
    Implements System.Web.IHttpHandler

    Sub ProcessRequest(ByVal context As HttpContext) Implements IHttpHandler.ProcessRequest

        ' pass an ID through the query string to append a unique identifier to your
        downloadable fileName

        Dim fileUniqueId As Integer = CInt(context.Request.QueryString("id"))

        ' file path could also be something like "C:\FolderName\FilesForUserToDownload

        Dim filePath As String = "\\ServerName\FolderName\FilesForUserToDownload"
        Dim fileName As String = "UserWillDownloadThisFile_" & fileUniqueId
        Dim fullFilePath = filePath & "\" & fileName
        Dim byteArray() As Byte = File.ReadAllBytes(fullFilePath)

        ' prompt the user to download the file

        context.Response.Clear()
        context.Response.ContentType = "application/x-please-download-me" ' "application/x-
unknown"
        context.Response.AppendHeader("Content-Disposition", "attachment; filename=" &
fileName)
        context.Response.BinaryWrite(byteArray)
        context.Response.Flush()
        context.Response.Close()
        byteArray = Nothing

    End Sub

    ReadOnly Property IsReusable() As Boolean Implements IHttpHandler.IsReusable
        Get
            Return False
        End Get
    End Property

End Class
```

Vous pouvez appeler le gestionnaire depuis le code derrière ou depuis une langue côté client. Dans cet exemple, j'utilise un javascript qui appellera le gestionnaire.

```
function openAttachmentDownloadHandler(fileId) {
```



```

// the location of your handler, and query strings to be passed to it

var url = "..\\_Handlers\\AttachmentDownload.ashx?";
url = url + "id=" + fileId;

// opening the handler will run its code, and it will close automatically
// when it is finished.

window.open(url);

}

```

Maintenant, associez la fonction javascript à un événement de clic sur un élément cliquable de votre formulaire Web. Par exemple:

```

<asp:LinkButton ID="lbtnDownloadFile" runat="server"
OnClientClick="openAttachmentDownloadHandler(20);">Download A File</asp:LinkButton>

```

Ou vous pouvez aussi appeler la fonction javascript à partir du code:

```

ScriptManager.RegisterStartupScript(Page,
    Page.GetType(),
    "openAttachmentDownloadHandler",
    "openAttachmentDownloadHandler(" & fileId & ");",
    True)

```

Lorsque vous cliquez sur votre bouton, le httpHandler envoie votre fichier au navigateur et demande à l'utilisateur s'il souhaite le télécharger.

Lire httpHandlers en ligne: <https://riptutorial.com/fr/asp-net/topic/3476/httphandlers>

Chapitre 18: Katana

Introduction

Qu'est-ce que Katana? Katana est un ensemble de composants open source permettant de créer et d'héberger des applications Web basées sur OWIN, gérées par Microsoft Open Technologies Group. . En outre, Katana fournit une grande variété de composants middleware prêts à l'emploi, prêts à être utilisés dans une application OWIN.

Exemples

Exemple

Application de base KatanaConsole

```
namespace KatanaConsole
{
    // use an alias for the OWIN AppFunc:
    using AppFunc = Func<IDictionary<string, object>, Task>;

    class Program
    {
        static void Main(string[] args)
        {
            WebApp.Start<Startup>("http://localhost:8080");
            Console.WriteLine("Server Started; Press enter to Quit");
            Console.ReadLine();
        }
    }

    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            var middleware = new Func<AppFunc, AppFunc>(MyMiddleWare);
            app.Use(middleware);
        }

        public AppFunc MyMiddleWare(AppFunc next)
        {
            AppFunc appFunc = async (IDictionary<string, object> environment) =>
            {
                // Do something with the incoming request:
                var response = environment["owin.ResponseBody"] as Stream;
                using (var writer = new StreamWriter(response))
                {
                    await writer.WriteAsync("<h1>Hello from My First Middleware</h1>");
                }
                // Call the next Middleware in the chain:
                await next.Invoke(environment);
            };
            return appFunc;
        }
    }
}
```



```
}  
}
```

Lire Katana en ligne: <https://riptutorial.com/fr/asp-net/topic/8236/katana>

Chapitre 19: Liaison de données

Exemples

Source de données SQL

Les contrôles pouvant être liés à des données peuvent utiliser des contrôles `SqlDataSource`. Le contrôle `SqlDataSource` vous permet non seulement d'extraire des données d'une base de données, mais également de modifier et de trier les données.

Récupération des données

Procédure stockée:

```
<asp:SqlDataSource ID="SqlDataSourceEmployees"
    runat="server"
    ConnectionString="<%= $ ConnectionStrings:MyConnectionString %>"
    SelectCommand="sp_GetEmployees"
    SelectCommandType="StoredProcedure">
</asp:SqlDataSource>
```

Requête SQL:

```
<asp:SqlDataSource ID="SqlDataSourceEmployees"
    runat="server"
    ConnectionString="<%= $ ConnectionStrings:MyConnectionString %>"
    SelectCommand="SELECT
        EmployeeID,
        EmployeeFirstName,
        EmployeeLastName
    FROM
        dbo.Employees">
</asp:SqlDataSource>
```

Paramètres:

```
<asp:SqlDataSource ID="SqlDataSourceEmployees"
    runat="server"
    ConnectionString="<%= $ ConnectionStrings:MyConnectionString %>"
    SelectCommand="SELECT
        EmployeeID,
        EmployeeFirstName,
        EmployeeLastName
    FROM
        dbo.Employees
    WHERE
        DepartmentID = @DepartmentID;">
<SelectParameters>
    <asp:ControlParameter ControlID="ddlDepartment"
        Name="DepartmentID"
        PropertyName="SelectedValue" />
</SelectParameters>
```



```
</SelectParameters>
</asp:SqlDataSource>
```

Soyez conscient de l'option `CancelSelectOnNullParameter` , qui, si elle est définie sur `true` (valeur par défaut), arrête la liaison de données si un paramètre est `NULL`.

Utilisation de base

GridView:

```
<asp:GridView ID="GridViewEmployees"
    runat="server"
    AutoGenerateColumns="false"
    DataSourceID="SqlDataSourceEmployees">
    <Columns>
        <asp:BoundField DataField="EmployeeID" HeaderText="Employee ID" />
        <asp:BoundField DataField="EmployeeFirstName" HeaderText="First Name" />
        <asp:BoundField DataField="EmployeeLastName" HeaderText="Last Name" />
    </Columns>
</asp:GridView>
```

Source de données d'objet

```
<asp:ObjectDataSource ID="ObjectDataSourceEmployees" runat="server"
    TypeName="MyPackage.MyDataAccessClass"
    DataObjectTypeName="MyPackage.Employee"
    SelectMethod="GetEmployees"
    UpdateMethod="SaveEmployee"
    InsertMethod="SaveEmployee">
</asp:ObjectDataSource>
```

Dans le code derrière

La classe d'accès aux données

```
public class MyDataAccess
{
    public static List<Employee> GetEmployees()
    {
        List<Employee> results = new List<Employee>()
        {
            new Employee(){ Id=1, Name="John Smith" },
            new Employee(){ Id=2, Name="Mary Jane" }
        };

        return results;
    }

    public static void SaveEmployee(Employee e)
    {
        // Persist Employee e to the DB/cache etc. here
    }
}
```


La classe des employés

```
public class Employee
{
    public Int32 EmployeeId { get; set; }
    public string Name { get; set; }
}
```

Lire Liaison de données en ligne: <https://riptutorial.com/fr/asp-net/topic/2245/liaison-de-donnees>

Chapitre 20: Liste de données

Syntaxe

1. **ItemTemplate** : il analyse le contenu et la disposition des éléments de la liste. Cette opération est obligatoire. Obligatoire
2. **AlternatingItemTemplate** : S'il est mentionné, détermine le contenu et la disposition des éléments en alternance. S'il n'est pas mentionné, ItemTemplate est utilisé.
3. **SeparatorTemplate** : Si mentionné, est rendu entre les éléments (et les éléments en alternance). Si ce n'est pas mentionné, un séparateur n'est pas rendu.
4. **SelectedItemTemplate** : S'il est mentionné, détermine le contenu et la disposition de l'élément sélectionné. S'il n'est pas mentionné, ItemTemplate (AlternatingItemTemplate) est utilisé.
5. **EditItemTemplate** : S'il est mentionné, détermine le contenu et la disposition de l'élément en cours de modification. S'il n'est pas mentionné, ItemTemplate (AlternatingItemTemplate, SelectedItemTemplate) est utilisé.
6. **HeaderTemplate** : S'il est mentionné, détermine le contenu et la disposition de l'en-tête de liste. Si non mentionné, l'en-tête n'est pas rendu.
7. **FooterTemplate** : S'il est mentionné, détermine le contenu et la disposition du pied de liste. S'il n'est pas mentionné, le pied de page n'est pas rendu.

Exemples

Liaison de données dans asp.net

Aspx

```
<asp:DataList runat="server" CssClass="sample" RepeatLayout="Flow" ID="dlsamplecontent"
RepeatDirection="Vertical" OnItemCommand="dlsamplecontent_ItemCommand">
    <ItemStyle CssClass="tdContainer" />
    <ItemTemplate>
        //you code
    </ItemTemplate>
</asp:DataList>
```

Aspx.cs

```
public void GetSamplingContentType()
{
    try
    {
        ErrorLogger.gstrClientMethodName = this.GetType().FullName + "_" +
        System.Reflection.MethodBase.GetCurrentMethod().Name + " : ";

        DataTable dt = new DataTable();
        dlsamplecontent.DataSource = dt;
        dlsamplecontent.DataBind();
    }
}
```



```
    }  
    catch (Exception ex)  
    {  
        ErrorLogger.ClientErrorLogger(ex);  
    }  
}
```

Commande d'élément et récupération de l'ID à l'aide d'un argument de commande

```
protected void dlsamplecontent_ItemCommand(object source, DataListCommandEventArgs e)  
{  
  
    try  
    {  
        int BlogId = Convert.ToInt32(e.CommandArgument.ToString());  
        if (e.CommandName == "SampleName")  
        {  
            //your code  
  
        }  
    }  
    catch (Exception ex)  
    {  
        ErrorLogger.ClientErrorLogger(ex);  
    }  
}
```

Lire Liste de données en ligne: <https://riptutorial.com/fr/asp-net/topic/7041/liste-de-donnees>

Chapitre 21: Méthodes de page

Paramètres

Paramètre	Détail
limite	Le paramètre de la méthode C #. Vous fournissez l'argument via la méthode de la page.
onSuccess	La fonction JavaScript exécutée lorsque l'appel de la méthode de page a réussi.
onError	La fonction JavaScript exécutée en cas d'erreur dans l'appel à la méthode de page.

Remarques

Plus d'un paramètre

Dans l'exemple, la fonction C # demande simplement un paramètre, si vous devez en transmettre plus d'un, vous pouvez le faire, mettez-les simplement dans votre appel JS et vous êtes prêt à partir. Ej.

```
//C#
public static int SumValues(int num1, int num2, int num3, ..., int numN)

//JS
PageMethods.SumValues(num1, num2, num3, ..., numN, onSuccess, onError);
```

Valeur de retour

Dans la fonction `onSuccess` , le résultat sera la valeur de retour de la fonction C #. Dans la fonction `onError` , le résultat sera l'erreur.

Exemples

Comment l'appeler

Ajoutez simplement le décorateur `using` au début et le `[WebMethod]` à la méthode `static` à appeler dans la page `aspx`:

```
using System.Web.Services;
```



```
public partial class MyPage : System.Web.UI.Page
{
    [WebMethod]
    public static int GetRandomNumberLessThan(int limit)
    {
        var r = new Random();
        return r.Next(limit);
    }
}
```

Dans votre fichier .aspx, ajoutez un asp: ScriptManager permettant les méthodes de page:

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

Ensuite, vous pouvez l'appeler de la manière suivante:

```
var limit= 42 // your parameter value
PageMethods.GetRandomNumberLessThan(limit, onSuccess, onError);
function onSuccess(result) {
    var randomNumber = result;
    // use randomNumber...
}
function onError(result) {
    alert('Error: ' + result);
}
```

Lire Méthodes de page en ligne: <https://riptutorial.com/fr/asp-net/topic/1411/methodes-de-page>

Chapitre 22: Middleware

Paramètres

Paramètre	Détails
<code>IDictionary<string, object></code> environment	C'est la seule collection dans laquelle OWIN communique des informations lors d'un appel. Toutes les clés peuvent être trouvées sur https://docs.asp.net/en/latest/fundamentals/owin.html#owin-keys

Remarques

Le type `AppFunc` n'est qu'un alias pour le type `Func<IDictionary<string, object>, Task>` pour raccourcir les signatures de méthode, un peu comme `typedef` en C++.

Exemples

Affiche le chemin de la requête et le temps nécessaire pour le traiter

```
//define a short alias to avoid chubby method signatures
using AppFunc = Func<IDictionary<string, object>, Task>;

class RequestTimeMiddleware
{
    private AppFunc _next;

    public RequestTimeMiddleware(AppFunc next)
    {
        _next = next;
    }

    public async Task Invoke(IDictionary<string, object> environment)
    {
        IOwinContext context = new OwinContext(environment);

        var path = context.Request.Path;
        var sw = Stopwatch.StartNew();
        //Queue up the next middleware in the pipeline
        await _next(environment);
        //When the request comes back, log the elapsed time
        Console.WriteLine($"Request for {path} processed in {sw.ElapsedMilliseconds}ms");
    }
}

public static class RequestTimeMiddlewareExtensions
{
    //Extension method as syntactic sugar, to get a meaningful way
    //in adding the middleware to the pipeline
    public static void UseRequestTimeMiddleware(this IAppBuilder app)
```



```
    {
        app.Use<RequestTimeMiddleware>();
    }
}

public class Startup
{
    public void Configuration(IApplicationBuilder app)
    {
        //add the Middleware as early as possible
        app.UseRequestTimeMiddleware();
        //Queue up every other module
        app.Use(async (environment, next) =>
        {
            await environment.Response.WriteAsync("Hello from the console world");
            await next();
        });
    }
}
```

Lire Middleware en ligne: <https://riptutorial.com/fr/asp-net/topic/6607/middleware>

Chapitre 23: Mise en cache ASP.NET

Exemples

Cache de données

ASP.Net expose l'API de cache pour stocker les données dans le cache pour les récupérer ultérieurement.

Commencer

Store string

```
Cache["key"]="value";
```

Récupérer une chaîne

```
var value="";
if (Cache["key"] != null)
    value = Cache["key"].ToString();
```

Vous pouvez également utiliser les méthodes **Ajouter** ou **Insérer** .

```
protected void Page_Load( object sender, EventArgs e)
{
    if ( this.IsPostBack )
    {
        label1.Text + = "Page is posted back";
    }
    else
    {
        label1.Text + = "Page is created";
    }

    if ( Cache [ "item" ] == null )
    {
        label1.Text + = "New item is created";
        DateTime item = DateTime.Now;
        label1.Text + = "Item is stored";
        Cache.Insert ( "item", item, null );
        DateTime.Now.AddSeconds ( 20 ), TimeSpan.Zero;
    }

    else
    {
        label1.Text + = "Item is accesses";
        DateTime item = ( DateTime) Cache [ "item" ];
        label1.Text + = "Time is: " + item.ToString();
        label1.Text + = <br/>";
    }

    label1.Text + = "<br/>";
}
```


Lire Mise en cache ASP.NET en ligne: <https://riptutorial.com/fr/asp-net/topic/9148/mise-en-cache-asp-net>

Chapitre 24: Planificateur DayPilot

Paramètres

Paramètre	Desc
DataStartField	spécifie la colonne de la source de données contenant le début de l'événement (DateTime)
DataStartField	spécifie la colonne de la source de données contenant le début de l'événement (DateTime)
DataEndField	spécifie la colonne de source de données qui contient la fin de l'événement (DateTime)
DataTextField	spécifie la colonne de données contenant le texte de l'événement (chaîne)
DataIdField	spécifie la colonne de source de données qui contient l'ID d'événement (chaîne ou entier)
DataResourceField	spécifie la colonne de données contenant la clé étrangère de la ressource d'événement (chaîne)

Remarques

Ce sont les bases de l'ordonnancement DayPilot qui doit être approfondi.

Exemples

Informations de base

Le widget Planificateur DayPilot affiche une ligne de temps pour plusieurs ressources. Prend en charge AJAX et HTML5. Localisation automatique et manuelle. Prise en charge complète du style CSS

Déclaration

```
<%@ Register Assembly="DayPilot" Namespace="DayPilot.Web.Ui" TagPrefix="DayPilot" %>
<DayPilot:DayPilotScheduler
  ID="DayPilotScheduler1"
  runat="server"

  DataStartField="eventstart"
  DataEndField="eventend"
  DataTextField="name"
```



```
DataIdField="id"
DataResourceField="resource_id"

CellGroupBy="Month"
Scale="Day"

EventMoveHandling="CallBack"
OnEventMove="DayPilotScheduler1_EventMove" >

</DayPilot:DayPilotScheduler>
```

Lire Planificateur DayPilot en ligne: <https://riptutorial.com/fr/asp-net/topic/6027/planificateur-daypilot>

Chapitre 25: Rechercher le contrôle par ID

Syntaxe

```
1. control.FindControl("Id Of The Control To Be Found")
```

Remarques

- `FindControl` n'est pas récursif, il ne recherche que les enfants immédiats du contrôle
- Il y a une surcharge `FindControl(String, int)` qui n'est pas en retrait pour un usage public
- Si rien n'est trouvé, `FindControl` renvoie `null`, c'est donc souvent une bonne idée de vérifier le résultat pour ne pas être `null`

Exemples

Accéder au contrôle TextBox dans la page aspx

```
TextBox txt = (TextBox)FindControl(yourtxt_Id);
```

Trouvez un contrôle dans un GridView, un répéteur, une liste, etc.

Si le contrôle a des lignes.

```
TextBox tb = GridView1.Rows[i].FindControl("TextBox1") as TextBox;
```

Ou s'il a des articles.

```
TextBox tb = Repeater1.Items[i].FindControl("TextBox1") as TextBox;
```

Lire Rechercher le contrôle par ID en ligne: <https://riptutorial.com/fr/asp-net/topic/6894/rechercher-le-contrôle-par-id>

Chapitre 26: Répétiteur

Exemples

Utilisation de base

Cet exemple crée un répétiteur simple à 1 colonne qui affiche une liste de nombres, un par élément de répétiteur.

Balisage:

```
<asp:Repeater ID="Repeater1" runat="server">
  <ItemTemplate>
    <%# Container.DataItem.ToString() %>
  </ItemTemplate>
</Repeater>
```

Code derrière:

```
protected void Page_Load(object sender, EventArgs e)
{
    List<int> numbers = new List<int>{1, 2, 3, 4, 5};
    Repeater1.DataSource = numbers;
    Repeater1.DataBind();
}
```

Lire Répétiteur en ligne: <https://riptutorial.com/fr/asp-net/topic/2635/repetiteur>

Chapitre 27: ScriptManager

Introduction

Le contrôle ScriptManager enregistre le script de la bibliothèque Microsoft AJAX avec la page. Cela permet des fonctionnalités de prise en charge de scripts clients telles que le rendu de pages partielles et les appels de services Web.

Syntaxe

1. <asp: ID de ScriptManager = "smPop" runat = "serveur"> </ asp: ScriptManager>
2. ScriptManager.RegisterStartupScript (Control, Type, String, String, Boolean);

Exemples

Travailler avec ScriptManager

Vous devez utiliser un contrôle ScriptManager sur une page pour activer les fonctionnalités suivantes d'ASP.NET AJAX:

1. Fonctionnalité de script client de la bibliothèque Microsoft AJAX et de tout script personnalisé que vous souhaitez envoyer au navigateur.

```
protected void Button1_Click(object sender, EventArgs e)
{
    Page.ClientScript.RegisterStartupScript(
        this.GetType(), "myscript", "alert('hello world!');");
}
```

2. Rendu de page partiel, qui permet de rafraîchir indépendamment les régions de la page sans publication. Les contrôles ASP.NET AJAX UpdatePanel, UpdateProgress et Timer nécessitent un contrôle ScriptManager pour prendre en charge le rendu de pages partielles.

3. **Les** classes de proxy JavaScript pour les services Web, qui vous permettent d'utiliser le script client pour accéder aux services Web en exposant les services Web en tant qu'objets fortement typés.

```
[WebMethod]
public int Add(int a, int b) { return a + b; }

function CallAdd()
{
    // method will return immediately
    // processing done asynchronously
    WebService.Add(0, 6, OnMethodSucceeded, OnMethodFailed);
}
```


4. Classes JavaScript pour accéder à l'authentification ASP.NET et aux services d'application de profil.

```
Sys.Services.AuthenticationService.login  
Sys.Services.AuthenticationService.logout  
  
<script type="text/javascript">  
    function MyMethod(username, password)  
    {  
        Sys.Services.AuthenticationService.login(username,  
            password, false, null, null, null, null, "User Context");  
    }  
</script>
```

Plus d'informations sur <https://msdn.microsoft.com/en-us/library/system.web.ui.scriptmanager.aspx>

Lire ScriptManager en ligne: <https://riptutorial.com/fr/asp-net/topic/10077/scriptmanager>

Chapitre 28: sections web.config> system.webServer / httpErrors & system.web / customErrors

Introduction

CustomErrors est un élément hérité (rétrocompatible), utilisé par Visual Studio Development Server (également appelé VSDS ou Cassini).

httpErrors est le nouvel élément utilisé uniquement par IIS7.

Exemples

Quelle est la différence entre customErrors et httpErrors?

Les deux sont utilisés pour définir la gestion des erreurs pour un site Web, mais différents logiciels font référence à différents éléments de configuration.

customErrors est un élément hérité (rétrocompatible), utilisé par Visual Studio Development Server (également appelé VSDS ou Cassini).

httpErrors est le nouvel élément utilisé uniquement par IIS7.

Cela met en évidence le problème possible lors du développement de sites Web ASP.NET lors de l'utilisation de VSDS au lieu du service IIS local.

Reportez- [vous également à ce message](#) pour savoir comment gérer les messages d'erreur avec IIS7, si vous souhaitez avoir le contrôle total de la sortie d'erreur.

Résumé:

1. Développement dans VSDS - utilisez customErrors
2. Publication du site sur IIS6 - utilisez customErrors
3. Publication du site sur IIS7 - utilisez httpErrors.
4. et si vous développez avec VSDS mais publiez dans IIS7, alors je suppose que vous aurez besoin des deux.

Lire sections web.config> system.webServer / httpErrors & system.web / customErrors en ligne:
<https://riptutorial.com/fr/asp-net/topic/10103/sections-web-config-gt--system-webserver---httperrors--amp--system-web---customerrors>

Chapitre 29: UpdatePanel

Introduction

Cette rubrique explique comment ajouter un support de mise à jour de page partielle à une page Web à l'aide de deux contrôles serveur Microsoft Ajax: le contrôle ScriptManager et le contrôle UpdatePanel. Ces contrôles suppriment la nécessité d'actualiser toute la page avec chaque publication, ce qui améliore l'expérience utilisateur.

Syntaxe

- `<asp:UpdatePanel ID = "UpdatePanel1" runat = "serveur">`
`</asp:UpdatePanel>`

Remarques

Un scriptManager doit être ajouté à la page pour que UpdatePanel fonctionne.

Exemples

Mise à jour de l'exemple de panneau

Étape 1: Ajouter ScriptManager à votre page

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

Étape 2: Ajoutez UpdatePanel à votre page juste après ScriptManager.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
  <ContentTemplate></ContentTemplate>
</asp:UpdatePanel>
```

Étape 3: Après avoir ajouté du contenu à votre modèle de contenu UpdatePanels, votre page aspx devrait ressembler à ceci:

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Untitled Page</title>
  <style type="text/css">
    #UpdatePanel1 {
      width:300px; height:100px;
```



```

    }
</style>
</head>
<body>
    <form id="form1" runat="server">
    <div style="padding-top: 10px">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <asp:UpdatePanel ID="UpdatePanel1" runat="server">
            <ContentTemplate>
                <fieldset>
                <legend>UpdatePanel</legend>
                <asp:Label ID="Label1" runat="server" Text="Panel created."></asp:Label><br />
                <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Button"
            />
            </fieldset>
        </ContentTemplate>
    </asp:UpdatePanel>
    <br />
    </div>
    </form>
</body>
</html>

```

Étape 4: Ajoutez cette partie à votre page C #:

```

protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text = "Refreshed at " +
        DateTime.Now.ToString();
}

```

Étape 5: Exécutez maintenant votre application.

Résultat attendu:

Le contenu du panneau change à chaque fois que vous cliquez sur le bouton, mais la page entière n'est pas actualisée. Par défaut, la propriété `ChildrenAsTriggers` d'un contrôle `UpdatePanel` est vraie. Lorsque cette propriété est définie sur `true`, les contrôles à l'intérieur du panneau participent aux mises à jour de pages partielles lorsqu'un contrôle dans le panneau provoque une publication.

Lire `UpdatePanel` en ligne: <https://riptutorial.com/fr/asp-net/topic/10075/updatepanel>

Chapitre 30: WebForms

Syntaxe

- `<asp: TextBox runat = "serveur" ID = "" TextMode = "" Text = "" />`
- `<asp: Repeater runat = "server" ID = "" OnItemDataBound = "">`
`<HeaderTemplate></HeaderTemplate>`
`<ItemTemplate></ItemTemplate>`
`<FooterTemplate></FooterTemplate>`
`</asp:Repeater>`

Remarques

Tous les contrôles ASP.Net WebForm requièrent `runat="server"` pour pouvoir communiquer avec CodeBehind.

Exemples

Utilisation d'un répéteur pour créer une table HTML

Lorsque le répéteur est lié, une nouvelle ligne de tableau sera ajoutée pour chaque élément des données.

```
<asp:Repeater ID="repeaterID" runat="server" OnItemDataBound="repeaterID_ItemDataBound">
  <HeaderTemplate>
    <table>
      <thead>
        <tr>
          <th style="width: 10%">Column 1 Header</th>
          <th style="width: 30%">Column 2 Header</th>
          <th style="width: 30%">Column 3 Header</th>
          <th style="width: 30%">Column 4 Header</th>
        </tr>
      </thead>
    </HeaderTemplate>
    <ItemTemplate>
      <tr runat="server" id="rowID">
        <td>
          <asp:Label runat="server" ID="mylabel">You can add ASP labels if you
want</asp:Label>
        </td>
        <td>
          <label>Or you can add HTML labels.</label>
        </td>
        <td>
          You can also just type plain text like this.
        </td>
        <td>
          <button type="button">You can even add a button to the table if you
want!</button>
        </td>
      </tr>
```



```

</ItemTemplate>
<FooterTemplate>
    </table>
</FooterTemplate>
</asp:Repeater>

```

La méthode `ItemDataBound` est facultative, mais utile pour formater ou remplir des données plus compliquées. Dans cet exemple, la méthode est utilisée pour donner dynamiquement à chaque `<tr>` un identifiant unique. Cet identifiant peut alors être utilisé en JavaScript pour accéder ou modifier une ligne spécifique. Notez que le `tr` ne conservera pas sa valeur d'ID dynamique sur `PostBack`. Le texte de chaque ligne `<asp:Label>` a également été défini dans cette méthode.

```

protected void repeaterID_ItemDataBound(object sender, RepeaterItemEventArgs e)
{
    if (e.Item.ItemType == ListItemType.Item || e.Item.ItemType ==
        ListItemType.AlternatingItem)
    {
        MyItem item = (MyItem)e.Item.DataItem;

        var row = e.Item.FindControl("rowID");
        row.ClientIDMode = ClientIDMode.Static;
        row.ID = "rowID" + item.ID;

        Label mylabel = (Label)e.Item.FindControl("mylabel");
        mylabel.Text = "The item ID is: " + item.ID;
    }
}

```

Si vous envisagez de faire beaucoup de communication avec `CodeBehind`, vous pouvez envisager d'utiliser `GridView`. Les répéteurs, en général, ont moins de charge que `GridView` et, avec une manipulation de base des identifiants, peuvent effectuer les mêmes fonctions que `GridView`.

Grouper dans `ListView`

`asp:ListView` introduit dans ASP.NET WebForms Framework 3.5 est le plus flexible de tous les contrôles `DataPresentation` du framework. Un exemple de regroupement à l'aide de `ListView` (qui sera utile en tant que galerie d'images)

Objectif : afficher trois images dans une ligne en utilisant `asp:ListView`

Balilage

```

<asp:ListView ID="SportsImageList" runat="server"
    GroupItemCount="3">
    <LayoutTemplate>
        <span class="images-list">
            <ul id="groupPlaceholder" runat="server"></ul>
        </span>
    </LayoutTemplate>
    <GroupTemplate>
        <ul>
            <li id="itemPlaceholder" runat="server"></li>
        </ul>
    </GroupTemplate>

```



```
</GroupTemplate>
<ItemTemplate>
    <li>
        <img src='<%# Container.DataItem %>' />
    </li>
</ItemTemplate>
</asp:ListView>
```

Code Derrière

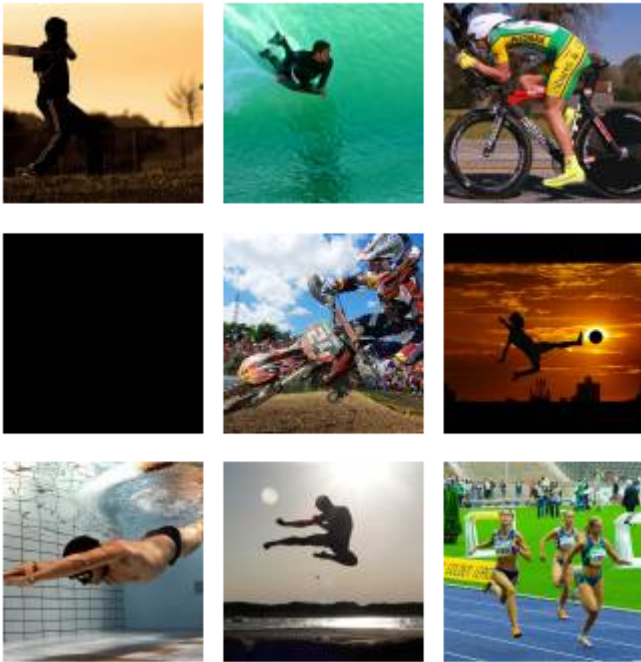
```
protected void Page_Load(object sender, EventArgs e)
{
    if(!IsPostBack)
    {
        SportsImageList.DataSource = GetImages();
        SportsImageList.DataBind();
    }
}

private static IEnumerable<string> GetImages()
{
    var images = Enumerable.Range(1, 9) //get numbers 1 to 9
        .Select(i =>
            string.Format("http://lorempixel.com/100/100/sports/{0}/", i)
        ); //convert the numbers to string
    return images;
}
```

CSS

```
.images-list ul{
    clear: both;
    list-style-type: none;
}
.images-list ul li{
    float: left;
    padding: 5px;
}
```

Sortie rendue



Exemple

```
<script language="VB" runat="server">

    Sub SubmitBtn_Click(sender As Object, e As EventArgs)
        Label1.Text = "Text1.Text = " & Text1.Text
    End Sub

</script>
```

```
<h3><font face="Verdana">TextBox Sample</font></h3>

<form runat="server">

    <asp:TextBox id="Text1" Text="Copy this text to the label" Width="200px" runat="server"/>

    <asp:Button OnClick="SubmitBtn_Click" Text="Copy Text to Label" Runat="server"/>

    <p>

    <asp:Label id="Label1" Text="Label1" runat="server"/>

</form>
```

Lien hypertexte

Le contrôle HyperLink permet de naviguer du client vers une autre page.

```
<html>

<script language="VB" runat="server">

    Sub Page_Load(sender As Object, e As EventArgs)
```



```
' Set hyperlink to "~", which indicates application root.
HyperLink1.NavigateUrl = "~"
End Sub

</script>

<body>

  <h3><font face="Verdana">Simple asp:hyperlink Sample</font></h3>

  <form runat=server>

    <p>

      <asp:hyperlink id=HyperLink1 runat="server">
        Go To QuickStart
      </asp:hyperlink>

    </p>

  </form>

</body>

</html>
```

Lire WebForms en ligne: <https://riptutorial.com/fr/asp-net/topic/5394/webforms>

Chapitre 31: WebService sans Visual Studio

Introduction

Un exemple ASP.Net très simple du strict minimum de code pour créer un WebService.

Remarques

Dans un article séparé de la documentation StackOverflow, nous examinerons comment utiliser ce service Web Calculator.

Exemples

Calculatrice WebService

```
<%@ WebService Language="C#" Class="Util" %>
using System;
using System.Web.Services;

public class Util: WebService
{
    [WebMethod]
    public int CalculatorAdd(int operandA, int operandB)
    {
        return operandA + operandB;
    }

    [WebMethod]
    public int CalculatorSubtract(int operandA, int operandB)
    {
        return operandA - operandB;
    }

    [WebMethod]
    public long CalculatorMultiply(int operandA, int operandB)
    {
        return operandA * operandB;
    }

    [WebMethod]
    public long CalculatorDivide(int operandNumerator, int operandDenominator)
    {
        if (operandDenominator == 0)
            return System.Int64.MaxValue;    // Should really do better error handling overall
        & return an error
        else
            return operandNumerator / operandDenominator;
    }
}
```

Lire WebService sans Visual Studio en ligne: <https://riptutorial.com/fr/asp-net/topic/8859/web-service-sans-visual-studio>

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec ASP.NET	Ahmed Abdelhameed , Aristos , Community , demonplus , Dillie-O , Josh E , khawarPK , Marco , Matt , Muhammad Awais , Satinder singh , wintersolider
2	Afficher l'état	jignesh
3	Asp Web Forms Identity	tatigo
4	ASP.NET - Contrôles de base	khawarPK
5	ASP.NET - Contrôles utilisateur	Tetsuya Yamamoto
6	ASP.NET - Etat de gestion	khawarPK
7	ASP.NET - Validateurs	khawarPK
8	Contrôles Asp.net Ajax	Saurabh Srivastava
9	Cycle de vie de la page	Abdul , mbenegas , Srikar , VDWWD
10	Délégation d'événement	Webruster
11	Directives	khawarPK , Tot Zam
12	Etat de session	Luke Ryan , Naveen Gogineni , Nisarg Shah
13	Expressions	Ryan
14	Gestion de session	Jasmin Solanki
15	Gestion des événements	khawarPK , Tot Zam
16	GridView	Andrei , Asif.Ali , j.f. , Marco , Ritwik

17	httpHandlers	Taylor Brown
18	Katana	jignesh
19	Liaison de données	j.f. , Ryan
20	Liste de données	Webruster
21	Méthodes de page	Enrique Zavaleta , wazz , XIII
22	Middleware	Marco
23	Mise en cache ASP.NET	tatigo
24	Planificateur DayPilot	Abdul
25	Rechercher le contrôle par ID	Andrei , VDWWD , Webruster
26	Répétiteur	Andrei
27	ScriptManager	Naveen Gogineni
28	sections web.config> system.webServer / httpErrors & system.web / customErrors	Naveen Gogineni
29	UpdatePanel	Naveen Gogineni
30	WebForms	Big Fan , jignesh , naveen , Tot Zam
31	WebService sans Visual Studio	George 2.0 Hope