ASP.NET MVC : vues

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

1. Introduction
2. Création d'une vue
3. Razor : syntaxe
4. Envoyer et récupérer des attributs
5. Les modèles de vue

Les vues

Avant 2012, on avait le choix entre deux moteurs de vue

- ASPX (assez proche de JSP) : utilise l'extension `.aspx`
- **Razor** : utilise l'extension `.cshtml` (pour C# + HTML) ou `.vbhtml` (pour VB + HTML)

Les vues

Comment créer une vue ?

- Faire clic droit sur le répertoire `Home` situé dans `Views`, aller dans `Ajouter > Vue`
- Dans `ViewName:` saisir `Index`
- Dans `Template:` choisir `Empty (without model)`
- Décocher toutes les cases
- Cliquer sur `Add`

Les vues

Code généré pour `Index.cshtml`

```
@{
    Layout = null;
}

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>View</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```

Les vues

Pour appeler la vue depuis le contrôleur

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        return View();
    }
}
```

Les vues

Pour appeler la vue depuis le contrôleur

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        return View();
    }
}
```

Explication

- Le compilateur va chercher la vue dans le répertoire `Home` (nom du contrôleur)
- La vue recherchée doit porter le même nom que la méthode appelante

Les vues

Pour appeler une vue par son nom

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        return View("ViewName");
    }
}
```

Les vues

Pour appeler une vue qui n'est pas dans le répertoire associé au contrôleur

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        return View("~/Views/Other/Other.cshtml"
            );
    }
}
```

Les vues

Une instruction sur une seule ligne

```
Hello @ViewBag.id
```

Les vues

Une instruction sur une seule ligne

```
Hello @ViewBag.id
```

Un bloc d'instruction

```
@{
  var nom = "Wick";
  var prenom = "John";
  var nomComplet = prenom + " " + nom;
}
<p>Bonjour @nomComplet</p>
```

Avec le mot clé `var`, on peut déclarer des variables, tout comme en C#

Les vues

On peut aussi typer les variables

```
string ville = "Marseille";
int codePostal = 13;
```

On peut aussi utiliser les types objets comme `DateTime`...

Les vues

Pour tester si une chaîne de caractère contient une valeur d'un certain type

- `maChaine.IsX()` : retourne `true` si `maVariable` est de type `X`, `false` sinon.
- `X` peut être : `Int`, `Float`, `Bool`, `DateTime`, `Decimal` et `Empty`.

Les vues

Pour tester si une chaîne de caractère contient une valeur d'un certain type

- `maChaine.IsX()` : retourne `true` si `maVariable` est de type `X`, `false` sinon.
- `X` peut être : `Int`, `Float`, `Bool`, `DateTime`, `Decimal` et `Empty`.

Exemple

```
@{
    var x = "5";
    var result = x.IsInt();
    var y = "true";
    var resultat = y.IsBool();
}
<div>@result</div> @* affiche True *@
<div>@resultat</div> @* affiche True *@
```

Les vues

Comment faire pour les autres types ?

On peut utiliser `maChaine.Is<type>()`

Les vues

Comment faire pour les autres types ?

On peut utiliser `maChaine.Is<type>()`

Exemple

```
@{
    var x = "5";
    var result = x.Is<char>();
    var y = "true";
    var resultat = y.Is<short>();
}
<div>@result</div> @* affiche True *@
<div>@resultat</div> @* affiche False *@
```

Les vues

Les fonctions de conversion de chaîne de caractères

- `maVariable.AsX()` : permet de convertir le contenu de la chaîne de caractères `maVariable` en type `X`
- `X` peut être : `Int`, `Float`, `Bool`, `DateTime`, `Decimal` et `Empty`.

Les vues

Les fonctions de conversion de chaîne de caractères

- `maVariable.AsX()` : permet de convertir le contenu de la chaîne de caractères `maVariable` en type `X`
- `X` peut être : `Int`, `Float`, `Bool`, `DateTime`, `Decimal` et `Empty`.

Exemple : conversion d'une chaîne de caractère en booléen

```
@{
    String z = "True";
    bool booleen = z.AsBool();
}
<div> @booleen </div> @* affiche True *@
```

Les vues

Comment faire pour les autres types ?

On peut utiliser `maChaine.As<type>()`

Les vues

Comment faire pour les autres types ?

On peut utiliser `maChaine.As<type>()`

Exemple

```
@{
    String x = "c";
    char caractere = x.As<char>();
}
<div> @caractere </div>@* affiche c *@
@{
    String y = "2";
    short i = y.As<short>();
}
<div> @i </div>@* affiche 2 *@
```

Les vues

Un traitement conditionnel avec `if`

```
@{
    int nbr = 10;
    if (nbr % 2 == 0)
    {
        <p> @nbr est pair </p>
    }
}
```

Ou aussi

```
@{
    int nbr = 10;
}
@if (nbr % 2 == 0)
{
    <p> @nbr est pair </p>
}
```

Les vues

Un bloc `if ... else`

```
@{
    int nbr = 10;
    if (nbr % 2 == 0)
    {
        <p> @nbr est pair </p>
    }
    else
    {
        <p> @nbr est impair </p>
    }
}
```

Les vues

Exemple avec `switch ... case`

```
<ul>
    @{
        switch (nbr)
        {
            case 0: <li>zéro</li>; break;
            case 1: <li>un</li>;break;
            default: <li>autre</li>;break;
        }
    }
</ul>
```

Les vues

Exemple avec `switch ... case`

```
<ul>
    @{
        switch (nbr)
        {
            case 0: <li>zéro</li>; break;
            case 1: <li>un</li>;break;
            default: <li>autre</li>;break;
        }
    }
</ul>
```

Ou aussi

```
<ul>
    @switch (nbr)
    {
        case 0:  <li>zéro</li>; break;
        case 1:  <li>un</li>; break;
        default: <li>autre</li>; break;
    }
</ul>
```

Les vues

Une boucle `for`

```
@for(var i = 0; i < 10; i++)
    {<p> @i</p>}
```

Affiche à la ligne les chiffres de 0 à 9

Les vues

Une boucle `for`

```
@for(var i = 0; i < 10; i++)
    {<p> @i</p>}
```

Affiche à la ligne les chiffres de 0 à 9

Exemple avec `foreach`

```
    <ul>
        @{
            var list = new List<int>();
            list.Add(2);
            list.Add(1);
            list.Add(7);
            foreach (var x in list)
            {
                <li>@x</li>
            }
        }
     </ul>
```

Affiche sous forme d'une liste les chiffres 2, 1 et 7

Les vues

On peut aussi écrire le code précédent ainsi

```
@{
    var list = new List<int>();
    list.Add(2);
    list.Add(1);
    list.Add(7);
}
<ul>
    @foreach (var x in list)
    {
        <li>@x</li>
    }
</ul>
```

Les vues

Exemple avec `while`

```
@{
    var list = new List<int>();
    list.Add(2);
    list.Add(1);
    list.Add(7);
}
<ul>
    @{
        var j = 0;
        while (j < list.Count)
        {
            <li>@list[j]</li>
                j++;
        }
    }
</ul>
```

Les vues

Exemple avec les tableaux

```
@{
    int[] list = { 2, 1, 7 };
}
<ul>
    @{
        var j = 0;
        while (j < list.Length)
        {
            <li>@list[j]</li>
                j++;
        }
    }
</ul>
```

Les vues

Exemple avec `if ... else` **et** `while`

```
<ul>
    @{
        int[] list = { 2, 1, 7 };
        var j = 0;
        while (j < list.Length)
        {
            if (list[j] % 2 == 0)
            {
                <li>@list[j]</li>
            }
            else
            {
                <li>@j</li>
            }
            j++;
        }
    }
</ul>
```

Les vues

Pour mettre en commentaire un bloc : utiliser `@* ... *@`

```
@*@{
    var x = 3;
}*@
```

Les vues

Envoyer un attribut par le contrôleur

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        if (id != null)
        {
            ViewData["id"] = id;
            return View();
        }
        else return View("Error");
    }
}
```

Les vues

Récupérer un attribut dans la vue

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-
       width" />
    <title>View</title>
</head>
<body>
    Number @ViewData["id"]
</body>
</html>
```

Les vues

On peut aussi utiliser `ViewBag`

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int ? id)
    {
        if (id != null)
        {
            ViewBag.id = id;
            return View();
        }
        else return View("Error");
    }
}
```

Les vues

Récupérer un attribut dans la vue

```
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-
       width" />
    <title>View</title>
</head>
<body>
    Number  @ViewBag.id
</body>
</html>
```

Les vues

`ViewBag` Vs `ViewData`

- `ViewBag` est un objet, `ViewData` est un dictionnaire (clé,valeur).
- `ViewBag` est moins rapide que `ViewData`.
- Contrairement à `ViewData`, `ViewBag` ne nécessite pas une conversion de type lors d'une énumération.

Les vues

Déplaçons la liste précédente dans le contrôleur

```
public class HomeController : Controller
{
    // GET: Home
    public ActionResult Index(int? id)
    {
        var list = new List<int>();
        list.Add(2);
        list.Add(1);
        list.Add(7);
        ViewData["list"] = list;
        ViewBag.list = list;
                    return View();
    }
}
```

Les vues

Récupérer la liste dans la vue (cas d'un `ViewBag`)

```
<ul>
  @foreach (var elt in @ViewBag.list)
  {
     <li> @elt </li>
  }
</ul>
```

Récupérer la liste dans la vue (cas d'un `ViewData`)

```
<ul>
  @foreach (var elt in @ViewData["list"] as List<int>)
  {
    <li> @elt </li>
  }
</ul>
```

Il faut le convertir, sinon une erreur sera signalée

Les vues

On peut aussi utiliser `View` pour passer un Modèle

Créer une classe `Personne` dans `Models`

```
namespace FirstAspNet.Models
{
    public class Personne
    {
        public int Num { get; set; }
        public string Nom { get; set; }
        public string Prenom { get; set; }
    }
}
```

Les vues

Dans l'action `Index()` du contrôleur

```
Personne p = new Personne()
{
    Num = 100,
    Nom = "Wick",
    Prenom = "John"
};
return View(p);
```

Les vues

Dans l'action `Index()` du contrôleur

```
Personne p = new Personne()
{
    Num = 100,
    Nom = "Wick",
    Prenom = "John"
};
return View(p);
```

N'oublions pas

```
using FirstAspNet.Models;
```

Les vues

Dans la vue

```
<ul>
    @{
        <li> @Model.Num </li>
        <li> @Model.Nom </li>
        <li> @Model.Prenom </li>
    }
</ul>
```

En ajoutant `@model FirstAspNet.Models.Personne` au début du fichier `Index.cshtml`, on a l'auto-complétion

Les vues

`@ViewBag`, `@ViewData`, `@Model`... dans la même page, un peu trop ?

- Solution : utiliser les modèles de vue

Les vues

Étape 1 : créer un répertoire `ViewModels`

- Faire un clic droit sur le nom du projet dans l'`Explorateur de solutions`
- Aller dans `Ajouter > Nouveau dossier`
- Renommer le `ViewModels` et valider

Les vues

Étape 2 : créer une classe `HomeViewModel` dans `ViewModels`

```
namespace FirstAspNet.ViewModels
{
    public class HomeViewModel
    {
        public Personne Personne { get; set; }
        public List<int> Liste { get; set; }
        public int Id { get; set; }
    }
}
```

Les vues

Étape 2 : créer une classe `HomeViewModel` dans `ViewModels`

```
namespace FirstAspNet.ViewModels
{
    public class HomeViewModel
    {
        public Personne Personne { get; set; }
        public List<int> Liste { get; set; }
        public int Id { get; set; }
    }
}
```

N'oublions pas

```
using FirstAspNet.Models;
```

Étape 3 : modifier le contrôleur (plus de `ViewBag`, plus de `ViewData`)

```
public class HomeController : Controller
{
    public ActionResult Index(int ? id)
    {
        var liste = new List<int>();
        liste.Add(2);
        liste.Add(1);
        liste.Add(7);
        Personne p = new Personne()
        {
            Num = 100,
            Nom = "Wick",
            Prenom = "John"
        };
        HomeViewModel homeViewModel = new HomeViewModel()
        {
            Liste = liste,
            Id = id ?? 5,
            Personne = p
        };
        return View(homeViewModel);
    }
}
```

Étape 4 : modifier la vue

```
@model FirstAspNet.ViewModels.HomeViewModel
@{
    Layout = null;
}
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width" />
    <title> View </title>
  </head>
  <body>
    <p> @Model.Id </p>
    <p> @Model.Personne.Nom </p>
    @foreach (var elt in Model.Liste)
    {
        <p> @elt </p>
    }
  </body>
</html>
```

Les vues

On peut associer un modèle de vue à une vue au moment de la création

- Faire un clic droit sur un des répertoires de `Views`
- Aller dans `Ajouter > Vue`
- Dans `View Name:` , saisir le nom
- Dans `Template:` , choisir `Empty` (ne pas confondre avec `Empty (without model)`)
- Dans `Model class:` , choisir `HomeViewModel`
- Valider en cliquant sur `Add`

La vue générée avec la référence

```
@model FirstAspNet.ViewModels.HomeViewModel

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>New</title>
</head>
<body>
    <div>
    </div>
</body>
</html>
```