ASP.NET MVC : helpers

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

Les helpers

Les helpers, c'est quoi ?

- est une classe qui facilite et simplifie l'écriture du code dans les vues
- permet de construire les éléments HTML de la vue en fonction des attributs définis dans un modèle
- affecte également la valeur des éléments HTML aux propriétés du modèle lors de la soumission d'un formulaire

Les helpers

Pour cela

- créer un contrôleur `MVC 5 Controller Empty` appelé `PersonneController`
- créer une vue (`Empty`) `Index`, associé au modèle `Personne`, dans le répertoire `Personne` (associé au contrôleur `PersonneController`) situé dans `Views`
- vérifier la référence du modèle `Personne` dans la vue

```
@model FirstAspNet.Models.Personne
```

Les helpers

Contenu de `PersonneController`

```
namespace FirstAspNet.Controllers
{
    public class PersonneController : Controller
    {
        // GET: Personne
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Les helpers

Modifions le contenu de `PersonneController`

```
namespace FirstAspNet.Controllers
{
    [RoutePrefix("Personne")]
    public class PersonneController : Controller
    {
        [Route("Ajouter")]
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

Les helpers

Pour créer un lien

```
@Html.ActionLink("Ajouter une personne", "Ajouter")
```

L'équivalent en HTML

```
<a href="/Personne/Ajouter/">Ajouter une personne</a
   >
```

Les helpers

Pour ajouter une zone texte

```
@Html.TextBox("Nom", null)
```

L'équivalent en HTML

```
<input id="Nom" name="Nom" type="text" value="" />
```

Les helpers

Pour ajouter une zone texte

```
@Html.TextBox("Nom", null)
```

L'équivalent en HTML

```
<input id="Nom" name="Nom" type="text" value="" />
```

On peut aussi écrire en fonction d'une propriété du modèle

```
@Html.TextBoxFor(perso => perso.Nom)
```

Les helpers

On peut associer une classe CSS à la zone texte

```
@Html.TextBox("Nom", null, new { @class = "form-
    control" })
```

Les helpers

On peut associer une classe CSS à la zone texte

```
@Html.TextBox("Nom", null, new { @class = "form-
    control" })
```

On peut aussi récupérer la valeur passée par le contrôleur et l'afficher dans la zone de texte

```
@Html.TextBox("Nom", @Model.Nom)
```

Les helpers

Pour associer un label à un élément HTML

```
@Html.Label("Nom")
```

L'équivalent en HTML

```
<label for="Nom">Nom</label>
```

Les helpers

Pour associer un label à un élément HTML

```
@Html.Label("Nom")
```

L'équivalent en HTML

```
<label for="Nom">Nom</label>
```

Pour associer au label une valeur différente du nom de l'élément

```
@Html.Label("Nom", "Votre nom")
```

L'équivalent en HTML

```
<label for="Nom">Votre nom</label>
```

Les helpers

Autres éléments helpers

- `@Html.TextArea(...)` : pour les `textarea` HTML
- `@Html.CheckBox(...)` : pour les cases à cocher
- `@Html.RadioButton(...)` : pour les boutons radios
- `@Html.DropDownList(...)` : pour les listes déroulantes
- `@Html.Hidden(...)` : pour les champs cachés
- `@Html.Password(...)` : pour le type mot de passe

Pas de helpers pour le type `bouton` et `submit`

Les helpers

Pour déclarer le début d'un formulaire

```
@{Html.BeginForm("Ajouter", "Personne");}
```

L'équivalent en HTML

```
<form action="/Personne/Ajouter/" method="post">
```

Par défaut, la méthode HTTP est `post` pour les formulaires. On peut la changer ainsi :

Les helpers

Pour déclarer le début d'un formulaire

```
@{Html.BeginForm("Ajouter", "Personne");}
```

L'équivalent en HTML

```
<form action="/Personne/Ajouter/" method="post">
```

Par défaut, la méthode HTTP est `post` pour les formulaires. On peut la changer ainsi :

```
@{Html.BeginForm("Ajouter", "Personne", FormMethod.
   Get);}
```

Les helpers

Pour déclarer la fin du formulaire

```
@{Html.EndForm();}
```

L'équivalent en HTML

```
</form>
```

Les helpers

On peut simplifier la déclaration d'un formulaire

```
@using (Html.BeginForm("Ajouter", "Personne"))
{
    ...
}
```

Les helpers

Editor

- un outil qui permet la construction d'un formulaire en fonction de type d'attribut déclaré dans le modèle

Les helpers

Considérons le modèle `Personne` suivant :

```
public class Personne
{
    public int Num { get; set; }
    public string Nom { get; set; }
    public string Prenom { get; set; }
    public bool Sportif { get; set; }
}
```

Les helpers

Utilisation d'`Editor` pour la construction du formulaire

```
Num :       @Html.Editor("Num")
Nom :    @Html.Editor("Nom")
Prenom :          @Html.Editor("Prenom")
Sportif : @Html.Editor("Sportif")
```

Les helpers

Utilisation d'`Editor` pour la construction du formulaire

```
Num :       @Html.Editor("Num")
Nom :    @Html.Editor("Nom")
Prenom :          @Html.Editor("Prenom")
Sportif : @Html.Editor("Sportif")
```

Ou en utilisant la fonction lambda

```
Num :       @Html.EditorFor(p => p.Num)
Nom :    @Html.EditorFor(p => p.Nom)
Prenom :          @Html.EditorFor(p => p.Prenom)
Sportif : @Html.EditorFor(p => p.Sportif)
```

Les helpers

L'équivalent en HTML

```
Num : <input class="text-box single-line" data-val="
   true" data-val-number="Le champ Num doit etre un
   nombre." data-val-required="Le champ Num est
   requis." id="Num" name="Num" type="number" value=
   "" />
Nom :   <input class="text-box single-line" id="Nom"
    name="Nom" type="text" value="" />
Prenom : <input class="text-box single-line" id="
   Prenom" name="Prenom" type="text" value="" />
Sportif : <input class="check-box" data-val="true"
   data-val-required="Le champ Sportif est requis."
   id="Sportif" name="Sportif" type="checkbox" value
   ="true" /><input name="Sportif" type="hidden"
   value="false" />
```

Les helpers

`data-val-required`, `data-val-number`... seront détaillées dans la partie validation

Les helpers

Transformation

- un attribut de type `int` sera transformé en `number` de HTML
- un attribut de type `bool` sera transformé en `checkbox` de HTML
- un attribut de type `string` sera transformé en `text` de HTML
- un attribut de type `decimal, float` sera transformé en `text` de HTML

Les helpers

On peut aussi faire

```
@using (Html.BeginForm("Ajouter", "Personne"))
{
    @Html.DisplayNameFor(model => model.Num);
    @Html.Editor("Num", null);
    @Html.DisplayNameFor(model => model.Nom);
    @Html.Editor("Nom");
    @Html.DisplayNameFor(model => model.Prenom);
    @Html.Editor("Prenom");
    @Html.DisplayNameFor(model => model.Sportif);
    @Html.Editor("Sportif");
    <input type="submit" value="Ajouter" />
}
```

Les helpers

On peut aussi faire

```
@using (Html.BeginForm("Ajouter", "Personne"))
{
    @Html.DisplayNameFor(model => model.Num);
    @Html.Editor("Num", null);
    @Html.DisplayNameFor(model => model.Nom);
    @Html.Editor("Nom");
    @Html.DisplayNameFor(model => model.Prenom);
    @Html.Editor("Prenom");
    @Html.DisplayNameFor(model => model.Sportif);
    @Html.Editor("Sportif");
    <input type="submit" value="Ajouter" />
}
```

Ce formulaire sera envoyé à une action (`Ajouter`) du contrôleur (`PersonneController`) qui accepte les requêtes Http de type `post` et qui prend en paramètre un objet du modèle `Personne` (contenant les valeurs envoyées par le formulaire)

Les helpers

La méthode `Ajouter` du contrôleur `PersonneController`

```
[HttpPost]
[Route("Ajouter")]
public String Ajouter(Personne p)
{
    return $"Personne { p.Prenom } { p.Nom } a bien
       été ajoutée";
}
```

Les helpers

La méthode `Ajouter` du contrôleur `PersonneController`

```
[HttpPost]
[Route("Ajouter")]
public String Ajouter(Personne p)
{
    return $"Personne { p.Prenom } { p.Nom } a bien
       été ajoutée";
}
```

`[HttpPost]` : pour indiquer que cette action serait exécutée à la réception d'une requête Http de type `post`

Les helpers

La méthode `Ajouter` du contrôleur `PersonneController`

```
[HttpPost]
[Route("Ajouter")]
public String Ajouter(Personne p)
{
    return $"Personne { p.Prenom } { p.Nom } a bien
       été ajoutée";
}
```

`[HttpPost]` : pour indiquer que cette action serait exécutée à la réception d'une requête Http de type `post`

`(Personne p)` : pour récupérer les valeurs soumises du formulaire dans un objet de type `Personne`

Les helpers

Pour contrôler la saisie des utilisateurs

- Définir les contraintes sous forme de décorateur dans le modèle (les entités)
- Préparer les messages à afficher en cas de violation d'une contrainte
- Dans le contrôleur, tester que ces contraintes sont satisfaites avant d'envoyer les données à la base de données.

Les helpers

Ajoutons les décorateurs dans l'entité `Personne`

```
public class Personne
{
    [Range(0, 1000)]
    public int Num { get; set; }
    [Required]
    public string Nom { get; set; }
    [MinLength(2)]
    public string Prenom { get; set; }
    public bool Sportif { get; set; }
}
```

Les helpers

Préparer l'affichage de message de validation dans la vue

`Index.cshtml`

```
@using (Html.BeginForm("Ajouter", "Personne"))
{
    @Html.DisplayNameFor(model => model.Num);
    @Html.Editor("Num");<br />
    @Html.ValidationMessage("Num", "")
    @Html.DisplayNameFor(model => model.Nom);<br />
    @Html.Editor("Nom");<br />
    @Html.ValidationMessage("Nom", "")<br />
    @Html.DisplayNameFor(model => model.Prenom);
    @Html.Editor("Prenom");<br />
    @Html.ValidationMessage("Prenom", "")<br />
    @Html.DisplayNameFor(model => model.Sportif);
    @Html.Editor("Sportif");<br />
    <input type="submit" value="Ajouter" />
}
```

Les helpers

De la même manière, on peut aussi utiliser les expressions lambda avec `@Html.ValidationMessageFor()`

```
@using (Html.BeginForm("Ajouter", "Personne"))
{
    @Html.DisplayNameFor(model => model.Num);
    @Html.EditorFor(model => model.Num);<br />
    @Html.ValidationMessageFor(model => model.Num)
        ...
}
```

Les helpers

Dans le contrôleur

```
public ActionResult Index()
{
    return View();
}
[HttpPost]
public ActionResult Ajouter(Personne p)
{
    if (ModelState.IsValid)
    {
        // persister les données
        return RedirectToAction("Index");
    }
    // afficher la même vue avec les messages d'erreur et
       les valeurs saisies
    return View("Index",p);
}
```

Les helpers

Pour tester, aller sur la route `/Personne/Index`

- Si on saisit un numéro supérieur à 1000
- Ou si on ne saisit pas de nom
- Ou si on saisit une seule lettre pour le prénom
- **Alors un message d'erreur sera affiché (un message par défaut, qu'on n'a jamais préparé)**

Les helpers

Pour personnaliser les messages d'erreur

```
public class Personne
{
    [Range(0, 1000)]
    public int Num { get; set; }
    [Required(ErrorMessage = "Merci de saisir le nom")]
    public string Nom { get; set; }
    [MinLength(2, ErrorMessage = "Au moins deux caractères")]
    public string Prenom { get; set; }
    public bool Sportif { get; set; }
}
```

Les helpers

Pour personnaliser les messages d'erreur

```
public class Personne
{
    [Range(0, 1000)]
    public int Num { get; set; }
    [Required(ErrorMessage = "Merci de saisir le nom")]
    public string Nom { get; set; }
    [MinLength(2, ErrorMessage = "Au moins deux caractères")]
    public string Prenom { get; set; }
    public bool Sportif { get; set; }
}
```

Pour les décorateurs de validation, il faut utiliser l'espace de noms suivant :

```
using System.ComponentModel.DataAnnotations;
```

Les helpers

On peut aussi ajouter certains détails dans le message d'erreur

```
public class Personne
{
    [Range(0, 1000,ErrorMessage ="Le champ {0} doit être compris entre
       {1} et {2}")]
    public int Num { get; set; }
    [Required(ErrorMessage = "Merci de saisir le nom")]
    public string Nom { get; set; }
    [MinLength(2, ErrorMessage = "Au moins deux caractères")]
    public string Prenom { get; set; }
    public bool Sportif { get; set; }
}
```

Les helpers

On peut aussi ajouter certains détails dans le message d'erreur

```
public class Personne
{
    [Range(0, 1000,ErrorMessage ="Le champ {0} doit être compris entre
       {1} et {2}")]
    public int Num { get; set; }
    [Required(ErrorMessage = "Merci de saisir le nom")]
    public string Nom { get; set; }
    [MinLength(2, ErrorMessage = "Au moins deux caractères")]
    public string Prenom { get; set; }
    public bool Sportif { get; set; }
}
```

`Le champ {0} doit être compris entre {1} et {2}`

- `{0}` : fait référence au nom du champ (ici `Num`)
- `{1}` : fait référence à la borne inférieure de l'intervalle (ici `0`)
- `{2}` : fait référence à la borne supérieure de l'intervalle (ici `1000`)

Les helpers

Autres contraintes de validation de formulaire

- `CreditCard` : permet de spécifier que la valeur de cet attribut doit correspondre à un numéro de carte bancaire
- `EmailAddress` : permet de spécifier que la valeur de cet attribut doit correspondre à une adresse e-mail
- `MaxLength` : précise le nombre de caractère max que peut avoir cet attribut
- `RegularExpression()` : détermine en utilisant une expression régulière ce qu'un champ peut accepter (`[RegularExpression(@"^0[0-9]{9}$")]` permet de valider un numéro de téléphone français)
- ...

Les helpers

On peut aussi retourner le message d'erreur dans le contrôleur si le nom existe dans la base de données par exemple

```
[HttpPost]
public ActionResult Ajouter(Personne p)
{
    if (// tester si le nom existe dans la BD)
    {
        ModelState.AddModelError("Nom", "Ce nom existe
           dans la BD");
        return View("Index",p);
    }
    if (ModelState.IsValid )
    {
         // persister les données
         return RedirectToAction("Index");
    }
    return View("Index",p);
}
```

Les helpers

Les propriétés générées dans les vues

- `data-val` : contient `true` si le champ est soumis à validation
- `data-val-length` : contient le message à afficher si la contrainte sur la longueur est violée
- `data-val-length-max` : contient la longueur max qu'un champ peut avoir
- `data-val-required` : contient le message à afficher si le champs soumis est vide
- `data-valmsg-for` : précise le champ concerné (cet attribut est généré dans une balise `span`)
- `data-valmsg-replace` : contient `true` si le message de validation a été remplacé par un message personnalisé
- ...

Les helpers

Pour utiliser un affichage ordonné des erreurs, on peut utiliser `@Html.ValidationSummary`

```
@Html.ValidationSummary(false, "")
@using (Html.BeginForm("Ajouter", "Personne"))
{
    @Html.DisplayNameFor(model => model.Num);
    @Html.Editor("Num");<br />
    @Html.DisplayNameFor(model => model.Nom);<br />
    @Html.Editor("Nom");<br />
    @Html.DisplayNameFor(model => model.Prenom);
    @Html.Editor("Prenom");<br />
    @Html.DisplayNameFor(model => model.Sportif);
    @Html.Editor("Sportif");<br />
    <input type="submit" value="Ajouter" />
}
```