ASP.NET MVC : routage

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 routes

Introduction

- ASP.NET MVC génère un fichier de routage par défaut pour chaque application créée
- On peut modifier la route par défaut
- On peut aussi définir une nouvelle

Les routes

Contenu du RouteConfig

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "
       Index", id = UrlParameter.Optional}
);
```

Les routes

Contenu du RouteConfig

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "
       Index", id = UrlParameter.Optional}
);
```

- Si l'URL est composée de trois segments, le premier sera le nom du contrôleur, le deuxième sera l'action et le dernier sera le paramètre (donc, cette route autorise seulement un seul paramètre
- Des valeurs par défaut ont été précisées pour chaque segment si une valeur pour ce dernier n'a pas été renseignée (Attention à l'ordre des segments)

Les routes

Créons une nouvelle route

```
routes.MapRoute(
    name: "Second",
    url: "second/{msg}/{nbr}",
    defaults: new { controller = "Second", action = "
       Index", msg = UrlParameter.Optional, nbr=
       UrlParameter.Optional});
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index"
       , id = UrlParameter.Optional});
```

Les routes

Créons une nouvelle route

```
routes.MapRoute(
    name: "Second",
    url: "second/{msg}/{nbr}",
    defaults: new { controller = "Second", action = "
       Index", msg = UrlParameter.Optional, nbr=
       UrlParameter.Optional});
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index"
       , id = UrlParameter.Optional});
```

Tandis que la route `Default` renvoie vers plusieurs contrôleurs, la route `Second` dirige vers un seul contrôleur (qui est `SecondControleur`). Elle prend au plus deux paramètres. Le deuxième est de type numérique.

Les routes

Supposons que le contenu du `SecondController` est le suivant :

```
public class SecondController : Controller
{
    // GET: Second
    public String Index(string msg, int nbr)
    {
        string result = " ";
        for(int i = 0; i< nbr; i++)
        {
            result += msg + " ";
        }
        return result;
    }
}
```

Les routes

Exemple

- `/second/wick/3` affiche `wick wick wick`
- `/second?msg=wick&nbr=3` affiche `wick wick wick`
- `/second/3/wick` génère une erreur
- `/second?nbr=3&msg=wick` affiche `wick wick wick`

Attention à l'ordre des routes

Si on inverse les deux routes `Default` et `Second`, la route `/second/wick/3` déclenchera une erreur aussi.

Les routes

On peut définir des contraintes sur les paramètres de nos routes

- `nbr = @"\d+"` : pour indiquer que `nbr` est de type numérique (`\d` : entier, + : au moins un)
- si l'url est `/second/wick/john`, alors une erreur 404 sera affichée.
- Le compilateur commence par chercher la route dont le contrôleur est second et dont le deuxième paramètre est de type entier.
- Pas de correspondance avec la route `"second/{msg}/{nbr}"`, il passe donc à la route suivante dans `RouteConfig`.
- Il ne trouve aucune correspondance, il retourne donc une erreur 404.

Pour utiliser les expressions régulières dans la plateforme .NET, il faut utiliser le namespace `System.Text.RegularExpressions`

Les routes

Dans ce cas, il faut ajouter une section `constraints`

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "
       Index"},
    constraints: new { id = @"\d+"}
);
```

Les routes

Autres contraintes (exprimées avec les expressions régulières)

- `nbr = @"\d{4}"` : pour indiquer que `nbr` doit contenir 4 chiffres
- `*` : 0 ou plusieurs fois
- `?` : 0 ou 1 fois
- `^` : commence par
- `$` : se termine par
- `.` : n'importe quel caractère
- `()` : le groupe
- `{n,m}` : minimum `n` fois, maximum `m` fois
- `{n}` : `n` fois exactement
- `{n,}` : minimum `n` fois

Les routes

Autres contraintes (exprimées avec les expressions régulières)

- $\backslash d$: un chiffre
- $\backslash D$: tout sauf un chiffre
- $\backslash w$: un caractère alphanumérique
- $\backslash W$: tout sauf un caractère alphanumérique
- $\backslash t$: un caractère de tabulation
- $\backslash n$: un caractère de retour à la ligne
- ...

Les routes

Autres contraintes (exprimées avec les expressions régulières)

- `[a-z]` : toutes les lettres entre a et z
- `[abcd]` : a, b, c, ou d
- `[A-Za-z]` : une lettre en majuscule et une en minuscule
- `[^a-d]` : tout sauf a, b, c, et d
- ...

Pour utiliser un caractère réservé (^, $...) dans une expression régulière, il faut le précéder par \

Les routes

Pour indiquer que le nombre de paramètre est indéterminé, il faut ajouter *

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{*id}",
    defaults: new { controller = "Home", action = "
       Index", id = UrlParameter.Optional}
);
```

Les routes

Pour indiquer que le nombre de paramètre est indéterminé, il faut ajouter `*`

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{*id}",
    defaults: new { controller = "Home", action = "
       Index", id = UrlParameter.Optional}
);
```

Dans le contrôleur, il faut récupérer une chaîne de caractère nommée `id` contenant la suite de paramètres séparés par /

Les routes

Pour utiliser les décorateurs, il faut

- mettre en commentaire les routes définies dans `RouteConfig`
- activer le routage par décorateur

Les routes

Nouveau contenu de `RouteConfig`

```
public class RouteConfig
{
    public static void RegisterRoutes(
       RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*
           pathInfo}");

        // activer le routage par décorateur
        routes.MapMvcAttributeRoutes();
    }
}
```

Les routes

Pour définir une route `home/index`

```
public  HomeController : Controller
{
    [Route("home/index")]
    public ActionResult Index()
    {
        return View();
    }
}
```

Les routes

Pour définir une route avec un paramètre `home/index/{id}`

```
public  HomeController : Controller
{
    [Route("home/index/{id}")]
    public ActionResult Index(string id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

Dans la vue

```
@ViewBag.id
```

Ne pas recevoir une valeur pour ce paramètre déclenche une exception

Les routes

Pour éviter l'exception

```
public  HomeController : Controller
{
    [Route("home/index/{id?}")]
    public ActionResult Index(string id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

Les routes

Pour un paramètre de type entier

```
public  HomeController : Controller
{
    [Route("home/index/{id:int}")]
    public ActionResult Index(int id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

Les routes

Pour définir un préfixe pour toutes les routes d'un contrôleur

```
[RoutePrefix("home")]
public  HomeController : Controller
{
    [Route("index/{id:int}")]
    public ActionResult Index(int id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

Les routes

Pour définir un préfixe pour toutes les routes d'un contrôleur

```
[RoutePrefix("home")]
public  HomeController : Controller
{
    [Route("index/{id:int}")]
    public ActionResult Index(int id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

En saisissant `home/index/3` **dans la barre d'adresse, la méthode** `Index` **de** `HomeController` **sera exécutée**

Les routes

Pour détacher le préfixe de la route d'une méthode

```
[RoutePrefix("home")]
public  HomeController : Controller
{
    [Route("~index/{id:int}")]
    public ActionResult Index(int id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

Les routes

Pour détacher le préfixe de la route d'une méthode

```
[RoutePrefix("home")]
public  HomeController : Controller
{
    [Route("~index/{id:int}")]
    public ActionResult Index(int id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

En saisissant `home/index/3` **dans la barre d'adresse, on aura une erreur. La route** `index/3` **permet d'exécuter la méthode** `Index` **de** `HomeController`**.**

Les routes

Pour définir une valeur min pour le paramètre `id`

```
[RoutePrefix("home")]
public  HomeController : Controller
{
    [Route("~index/{id:int:min{5}}")]
    public ActionResult Index(int id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

Les routes

Pour définir une valeur min pour le paramètre `id`

```
[RoutePrefix("home")]
public  HomeController : Controller
{
    [Route("~index/{id:int:min{5}}")]
    public ActionResult Index(int id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

En saisissant `index/3` dans la barre d'adresse, on aura une erreur. La route `index/6` permet d'exécuter la méthode `Index` de `HomeController`.

Les routes

On peut aussi utiliser les fonctions

- `max`
- `minlength` et `maxlength`
- `range`
- `regex`
- ...

Les routes

Exemple avec les expressions régulières

```
[RoutePrefix("home")]
public  HomeController : Controller
{
    [Route("~/index/{id:regex(^0.*9$)}")]
    public ActionResult Index(int id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

Les routes

Exemple avec les expressions régulières

```
[RoutePrefix("home")]
public  HomeController : Controller
{
    [Route("~/index/{id:regex(^0.*9$)}")]
    public ActionResult Index(int id)
    {
        ViewBag.id = id;
        return View();
    }
}
```

Ainsi on impose une valeur pour le paramètre `id` qui commence par 0 et qui se termine par 9.