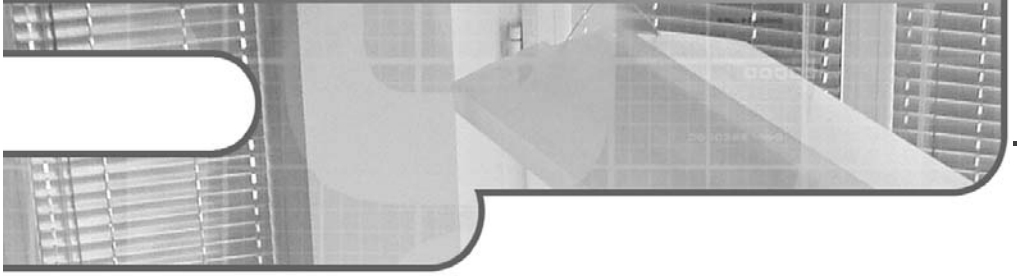


Chapitre 7
Utilisation avancée des contrôleurs

1. Introduction

Ce chapitre détaille le fonctionnement interne d'ASP.NET MVC concernant la résolution et l'invocation des contrôleurs. Ce chapitre poursuit l'étude du traitement d'une requête commencée dans le chapitre précédent, en reprenant à l'étape d'instanciation, par le module HTTP `UrlRoutingModule`, du gestionnaire HTTP par défaut d'ASP.NET MVC, de type `MvcHandler`, prenant en paramètre une instance de la classe `RequestContext`.

Une fois initié au cycle de vie d'un contrôleur, le lecteur trouvera des informations sur l'extensibilité des types de retour des actions, sur l'utilisation des filtres d'action et pour terminer, sur l'écriture de contrôleurs asynchrones.

2. Le cycle de vie d'un contrôleur

Lorsque la méthode `ProcessRequest`, point d'entrée d'un gestionnaire HTTP, est appelée au sein de la classe `MvcHandler`, le nom du contrôleur ciblé depuis l'instance `RequestContext` est tout d'abord extrait. Une méthode publique nommée `GetRequiredString`, exposée par la classe `RouteData`, est réservée à cet usage.

C'est à cette étape qu'un en-tête HTTP indiquant la version d'ASP.NET MVC est automatiquement ajouté à la réponse (sous la clé `X-AspNetMvc-Version`). Il est facile de désactiver cet en-tête car la classe `MvcHandler` contient une propriété statique booléenne appelée `DisableMvcResponseHeader`, dont la valeur par défaut est `false`. En définissant sa valeur à `true`, par exemple dans le fichier `Global.asax`, l'ajout de l'en-tête à la réponse sera alors omis.

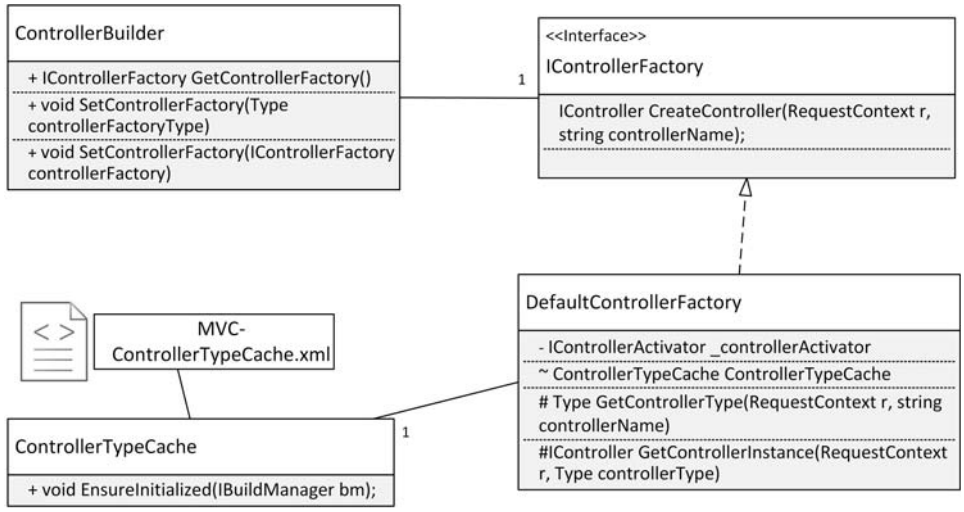
```
MvcHandler.DisableMvcResponseHeader = true;
```

Les problématiques soulevées par la présence de cet entête, donnant des informations sur la technologie utilisée, sont davantage expliquées dans le chapitre Sécurité.

2.1 La fabrication d'un contrôleur

Une fois le nom du contrôleur à instancier trouvé, le gestionnaire fait appel à la classe `ControllerBuilder`, disponible sous la forme du paradigme de programmation *Singleton*. Cette classe expose une instance qui doit implémenter l'interface `IControllerFactory` via la méthode `GetControllerFactory`. Elle permet également de remplacer le type exposé par défaut par une implémentation personnalisée via ses deux surcharges `SetControllerFactory`.

Le diagramme de classes suivant permettra au lecteur de conserver une vision d'ensemble pendant les explications qui suivent.



Remarque

Le fonctionnement interne de la classe `ControllerBuilder`, l'écriture d'une implémentation personnalisée de l'interface `IControllerFactory` et son utilisation pour la résolution des contrôleurs sont des thèmes étudiés dans le chapitre ASP.NET MVC avancé. Remplacer l'implémentation de l'interface `IControllerFactory` est un cas d'usage courant lorsqu'une libraire d'injection de dépendances est utilisée.

L'interface `IControllerFactory` représente la stratégie à utiliser pour générer une instance de contrôleur. Elle correspond au contrat ci-dessous.

```
public interface IControllerFactory
{
    IController CreateController(RequestContext requestContext,
string controllerName);
    SessionStateBehavior
GetControllerSessionBehavior(RequestContext requestContext
string controllerName);
    void ReleaseController(IController controller);
}
```

La classe utilisée par défaut par ASP.NET MVC et qui implémente l'interface `IControllerFactory` est la classe `DefaultControllerFactory` de l'espace de noms `System.Web.Mvc`.

Le but de la méthode `CreateController` est d'instancier et de retourner une implémentation de l'interface `IController` à partir des informations contenues dans une instance de `RequestContext` et du nom de contrôleur qui a été extrait depuis les paramètres de la route.

Étudier l'implémentation de cette méthode dans la classe `DefaultControllerFactory` est très instructif.

Outre les vérifications sur la validité des arguments qu'elle reçoit, un appel est d'abord fait à une méthode `GetControllerType`. Cette méthode extrait d'éventuelles informations sur l'espace de noms où le contrôleur doit être recherché, dans le cas d'une route pour une aire particulière. Puis, elle utilise les résultats obtenus pour appeler la méthode interne `GetControllerTypeWithinNamespaces`.

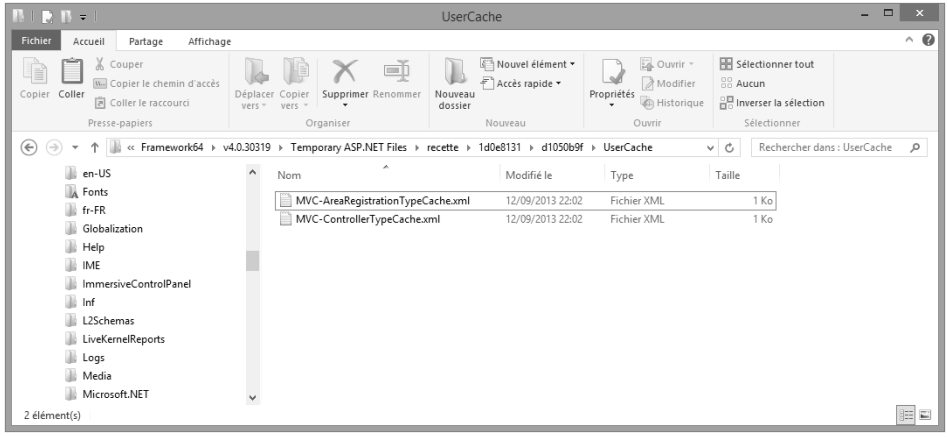
En interne, cette méthode utilise une instance de la classe `ControllerTypeCache`. Pendant tout le cycle de vie de l'application, cette classe maintient un cache de tous les types correspondant à des contrôleurs présents dans l'application. Ainsi, la recherche d'un contrôleur et son instanciation peut être accélérée puisque l'étape de réflexion n'est pas rejouée à chaque fois. Ce cache est constitué au premier appel de la méthode `EnsureInitialized`, qui a lieu à la première résolution d'un contrôleur. Par défaut, pour être localisé et pour pouvoir être instancié, un contrôleur doit respecter les règles suivantes :

– Sa visibilité doit être publique.
– Il doit implémenter l'interface `IController`. C'est le cas de la classe de base `Controller` mais aussi de la classe de base `ApiController`, utilisée avec Web API.
– Le nom de la classe doit être suffixé du terme *Controller*. C'est pour cette raison que la fenêtre de création d'un nouveau contrôleur propose automatiquement de l'ajouter.

Remarque

Les conventions de nommage ont un rôle important au sein d'ASP.NET MVC. Lors du chapitre sur la Conception de vues, une convention importante concernait la résolution des vues devant être nommées en fonction du nom des actions et placées dans un dossier du nom du contrôleur. Pour les contourner, les développeurs n'ont d'autres choix que de donner leur propre implémentation de ces fonctionnalités. Dans le cas présent, il faudrait implémenter l'interface `IControllerFactory`.

Le cache maintenu par la classe `ControllerTypeCache` est stocké dans un fichier XML nommé `MVC-ControllerTypeCache.xml` et qui est situé dans le répertoire temporaire du serveur Internet, dans un chemin du type : C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Temporary ASP.NET Files\front-release\d45c3dcb\e807656\UserCache.



L'exemple ci-dessous présente un extrait de ce fichier de cache. Il contient une liste d'assemblage avec leur nom complet, et pour chacun de ces assemblages, la liste des types qui ont été trouvés comme étant des contrôleurs utilisables.

```
<?xml version="1.0" encoding="utf-8"?>
<!--This file is automatically generated. Please do not modify
the contents of this file.-->
<typeCache lastModified="25/06/2013 19:35:47"
mvcVersionId="23916767-7e3a-4fab-ad85-0bb94082b2dd">
  <assembly name="MonAssembly, Version=1.0.0.0, Culture=neutral,
 PublicKeyToken=null">
    <module versionId="baba59b6-5c8b-43d2-a5d3-87f2bf656b8e">
      <type>Mes.Namespace.AccountController</type>
      <type>Mes.Namespace.HomeController</type>
    </module>
  </assembly>
</typeCache>
```

Remarque

Le même principe de mise en cache a lieu au lancement d'une application lorsque la méthode `AreaRegistration.RegisterAllAreas` est appelée. Celle-ci se contente de chercher des classes dérivant de la classe `AreaRegistration` et place son cache dans un fichier `MVC-AreaRegistrationTypeCache.xml` qui est sauvegardé au même endroit que celui des contrôleurs.

Si un seul type est trouvé, il est retourné à la méthode `CreateController`. Si aucun type n'est trouvé, c'est la valeur nulle qui est retournée, et c'est à la méthode `CreateController` de gérer ce cas d'erreur. Si à l'inverse, plus d'un contrôleur a été trouvé lors de la recherche, il y a ambigüité. C'est à ce moment que la fabrique de contrôleurs lève l'exception `CreateAmbiguousControllerException` en spécifiant tous les types qui ont été trouvés pour la route courante.