Module 4: Développement de contrôleurs ASP.NET MVC 5

# **Contenu:**

|  |  |
| --- | --- |
|  | [Aperçu du module](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C04.html#P1) |
| **Leçon 1:** | [Écriture de contrôleurs et d'actions](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C04.html#P2) |
| **Leçon 2:** | [Écriture de filtres d'action](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C04.html#P3) |
| **Laboratoire:** | [Développement de contrôleurs ASP.NET MVC 5](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C04.html#P4) |
|  | [Examen du module et points à retenir](https://www.skillpipe.com/api/2.1/content/urn:uuid:4ed7876c-8864-4cf5-91ee-491726dfe322@2020-12-11T17:05:09Z/OPS/html/20486C04.html#P5) |

# **Aperçu du module**

Les contrôleurs MVC répondent aux demandes du navigateur, créent des objets de modèle et les transmettent aux vues pour le rendu et l'affichage dans le navigateur Web. Si nécessaire, les contrôleurs peuvent également effectuer d'autres actions, telles que l'enregistrement des modifications de classe de modèle dans la base de données. Les contrôleurs sont au cœur des applications MVC. Vous devez comprendre le fonctionnement des contrôleurs pour pouvoir créer les bons objets de modèle, les manipuler et les transmettre aux bonnes vues.

Pour maximiser la réutilisation du code dans les contrôleurs, vous devez savoir comment programmer des filtres d'action. Vous pouvez utiliser des filtres d'action pour exécuter du code avant ou après chaque action de votre application Web, sur chaque action d'un contrôleur ou sur d'autres combinaisons d'actions de contrôleur.

### **Objectifs**

Après avoir terminé ce module, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Ajoutez un contrôleur à une application Web qui répond aux actions utilisateur spécifiées dans la conception du projet. |
| • | Écrivez du code dans des filtres d'action qui s'exécutent avant ou après une action du contrôleur. |

# Leçon 1: Écriture de contrôleurs et d'actions

Un contrôleur est une classe .NET Framework qui hérite de la classe de base System.Web.Mvc.Controller. Les contrôleurs répondent aux demandes des utilisateurs. Au sein d'une classe de contrôleurs, vous créez des actions pour répondre aux demandes des utilisateurs. Les actions sont des méthodes au sein d'un contrôleur qui renvoient un objet ActionResult. L'objet ActionResult est souvent une vue qui affiche une réponse à la demande de l'utilisateur; cependant, il peut également donner d'autres types de résultats. Pour traiter les demandes des utilisateurs entrants, gérer les entrées et les interactions des utilisateurs et implémenter la logique d'application appropriée, vous devez savoir comment créer des contrôleurs et des actions. Vous devez également savoir comment créer des paramètres dans des blocs de code d'action et transmettre des objets à des actions.

## **Objectifs de la leçon**

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Décrivez comment un contrôleur répond aux actions des utilisateurs dans une application Web MVC 5. |
| • | Écrivez des actions de contrôleur pour répondre aux demandes du navigateur Web, créer des classes de modèle et appeler des vues. |
| • | Expliquez comment utiliser les paramètres transmis dans une requête de navigateur à une action de contrôleur et les utiliser pour modifier le résultat de l'action. |
| • | Expliquez comment transmettre des informations aux vues qui ont des classes de modèle. |
| • | Créez un contrôleur et des actions. |
| • | Décrivez les usines de contrôleurs. |

## **Répondre aux demandes des utilisateurs**

Lorsqu'une application Web MVC reçoit une demande d'un navigateur Web, les événements suivants se produisent dans l'ordre:

|  |  |
| --- | --- |
| 1. | Un objet MvcHandler crée une fabrique de contrôleurs. La fabrique de contrôleurs est l'objet qui instancie un contrôleur pour répondre à la demande. Habituellement, cette fabrique est un objet DefaultControllerFactory, mais vous pouvez créer une fabrique de contrôleurs personnalisée, si nécessaire. L'objet MvcHandler choisit la fabrique de contrôleurs en fonction de l'objet RequestContext, qui contient des informations sur la demande effectuée par l'utilisateur. |
| 2. | La fabrique de contrôleurs crée un objet Controller et le MvcHandler appelle la méthode Execute dans ce contrôleur. |
| 3. | ControllerActionInvoker examine l'objet RequestContext et détermine l'action à appeler dans l'objet Controller. |
| 4. | ControllerActionInvoker utilise un classeur de modèle pour déterminer les valeurs à transmettre à l'action en tant que paramètres. |
| 5. | ControllerActionInvoker exécute l'action. Souvent, l'action crée une nouvelle instance d'une classe de modèle, peut-être en interrogeant la base de données avec les paramètres que l'invocateur lui a transmis. Cet objet de modèle est passé à une vue, pour afficher les résultats à l'utilisateur. Les méthodes d'action peuvent faire beaucoup d'autres choses telles que le rendu des vues et des vues partielles, la redirection vers d'autres sites Web, l'affichage d'extraits de contenu ou l'affichage d'autres fichiers. |

**Noter:**Le moteur de routage détermine quel contrôleur et quelle action reçoivent une demande. Le routage n'est pas couvert dans cette section.

**La demande de l'utilisateur**

Les utilisateurs de navigateurs Web font des demandes en saisissant une URL dans la barre d'adresse du navigateur ou en cliquant sur un lien vers une adresse de votre site Web. Ces liens peuvent se trouver soit dans votre site Web, auquel cas vous pouvez contrôler la façon dont ils sont rendus, soit à partir d'un site Web externe. Que la demande provienne du site Web ou d'un site Web externe, elle peut inclure des informations que les actions du contrôleur peuvent utiliser comme paramètres. Considérez les exemples suivants:

|  |  |
| --- | --- |
| • | http://www.adventure-works.com/: Cette URL est la page d'accueil du site Web et ne spécifie aucune information supplémentaire. |
| • | http://www.adventure-works.com/photo: Cette URL spécifie une valeur supplémentaire, photo. Par défaut, le MvcHandler interprète cela comme le nom d'un contrôleur. |
| • | http://www.adventure-works.com/photo/index: Cette URL spécifie une deuxième valeur, index. Par défaut, le MvcHandler interprète cela comme le nom d'une action dans le contrôleur. |
| • | http://www.adventure-works.com/photo/display/1: Cette URL spécifie une troisième valeur, 1. Par défaut, ControllerActionInvoker l'interprète comme un paramètre à transmettre à la méthode d'action. |
| • | http://www.adventure-works.com/photo/display?id=1: cette URL comprend une chaîne de requête, id = 1. Le classeur de modèles examine les actions d'affichage dans le contrôleur photo. S'il trouve une action avec un paramètre appelé id, il appelle cette action et passe 1 comme paramètre. |

**Noter:**Vous pouvez modifier la logique précédente de plusieurs manières. Par exemple, vous pouvez créer des itinéraires qui interprètent différemment les URL précédentes. Les exemples sont vrais lorsque seule la route par défaut existe.

Considérons qu'un utilisateur demande un contrôleur appelé photo, en tapant l'URL dans la barre d'adresse d'un navigateur Web. Par défaut, MVC DefaultControllerFactory nomme cette classe de contrôleur comme PhotoController. Vous devez vous en tenir à cette convention lorsque vous créez et nommez des contrôleurs. Sinon, vous recevrez des erreurs 404 inattendues et les contrôleurs ne fonctionneront pas comme prévu. Si vous créez une fabrique de contrôleurs personnalisée, vous pouvez définir votre propre convention de dénomination pour les classes de contrôleurs.

Les modèles de projet Microsoft Visual Studio incluent un dossier nommé Controllers. C'est un bon emplacement pour créer vos contrôleurs. Microsoft Visual Studio place les contrôleurs dans l'espace de noms projectname.Controllers, par défaut.

**Une classe de contrôleur simple**

public class PhotoController: Controller

{

contextDB privé = nouveau ContextDB ();

Index ActionResult public ()

{

return View ("Index", context.Photos.ToList ());

}

}

**Question:**Quelle est la convention à suivre lors de la création de contrôleurs?

## **Ecrire des actions de contrôleur**

Les contrôleurs encapsulent la logique d'interaction utilisateur pour une application Web MVC. Vous spécifiez cette logique en écrivant des actions. Une action est une méthode au sein de la classe de contrôleur. Le code que vous écrivez dans une méthode d'action détermine la manière dont le contrôleur répond à la demande, ainsi que la classe de modèle et la vue que MVC utilise pour afficher une page Web dans le navigateur.

**Noter:**Lorsque vous ajoutez un nouveau contrôleur à une application MVC, Visual Studio présente des options de génération de modèles pour vous aider à créer des méthodes d'action et des vues associées. Par exemple, si vous spécifiez une classe de modèle et une classe de contexte Entity Framework, Visual Studio peut créer un index et des détails de l'échafaudage, et créer, modifier et supprimer des méthodes d'action dans votre nouveau contrôleur. Vous pouvez les utiliser comme point de départ pour votre code. Au fur et à mesure que vous maîtriserez les méthodes d'action, vous préférerez peut-être sélectionner le modèle Empty MVC Controller et écrire vos propres méthodes sans code d'échafaudage.

Les actions de contrôleur sont des méthodes publiques qui renvoient un objet ActionResult. Les actions peuvent également renvoyer des objets de nombreuses autres classes qui dérivent de la classe ActionResult. Par exemple, vous pouvez écrire du code pour un contrôleur avec une action Index pour obtenir tous les objets Photo et les transmettre à la vue Index.

**Une action de détails**

public ActionResult First ()

{

Photo firstPhoto = context.Photos.ToList () [0];

if (firstPhoto! = null)

{

return View ("Détails", firstPhoto);

}

autre

{

return HttpNotFound ();

}

}

Certaines actions utilisateur sont en deux parties. Par exemple, pour créer une nouvelle photo, un utilisateur peut envoyer une requête HTTP GET à l'URL suivante: http://www.adventure-works.com/photo/create.

**Une action de création pour la demande GET**

public ActionResult Create ()

{

Photo newPhoto = nouvelle photo ();

return View ("Create", newPhoto)

}

La vue Créer affiche un formulaire dans lequel les utilisateurs peuvent remplir les détails de la photo, tels que le titre, la description, etc. Lorsqu'un utilisateur clique sur le bouton Soumettre, le navigateur Web effectue une requête HTTP POST.

**Une action de création pour la requête POST**

[HttpPost]

public ActionResult Create (Photo photo)

{

if (ModelState.IsValid)

{

context.Photos.Add (photo);

context.SaveChanges ();

return RedirectToAction ("Index");

}

autre

{

retour Vue ("Créer", photo);

}

}

Notez que la propriété ModelState.IsValid est utilisée pour vérifier si l'utilisateur a soumis des données valides. Vous pouvez spécifier la validation des données à l'aide des annotations de données de validation dans la classe de modèle. Si les données sont valides, l'objet modèle est ajouté et enregistré. Sinon, l'application affiche à nouveau la vue Créer, afin que l'utilisateur puisse corriger les données non valides.

**Noter:**Pour que le code précédent fonctionne, la vue Créer doit contenir un formulaire qui utilise la méthode POST.

**Classes de retour possibles**

Les méthodes d'action sont généralement définies avec la classe ActionResult comme type de retour. ActionResult est une classe de base et vous pouvez utiliser une plage de classes dérivées pour renvoyer différentes réponses au navigateur Web.

Les actions du contrôleur renvoient généralement une vue et lui transmettent une classe de modèle, pour affichage. Vous pouvez créer une action qui appelle l'assistant View () et crée et renvoie un objet ViewResult. L'assistant View () est disponible lorsque vous dérivez de la classe Controller de base.

Vous pouvez également renvoyer une erreur HTTP. Par exemple, vous pouvez créer une action de telle sorte que si un objet Photo est introuvable, le code crée une erreur 404 introuvable à l'aide de l'assistant HttpNotFound ().

Parfois, vous souhaiterez peut-être renvoyer un fichier à partir d'une méthode d'action. Par exemple, considérons que dans un modèle Photo, le fichier image est stocké sous forme de tableau d'octets dans la base de données. Pour afficher ce tableau d'octets en tant qu'image sur une page Web, l'action doit le renvoyer sous la forme d'un fichier .jpeg, qui peut être utilisé pour l'attribut src d'une balise HTML <img>. Pour renvoyer des fichiers, vous pouvez utiliser l'assistant File () pour renvoyer un objet FileContentResult. Vous pouvez utiliser cette technique dans l'action GetImage.

Les autres résultats possibles de l'action comprennent:

|  |  |
| --- | --- |
| • | PartialViewResult. Vous pouvez utiliser cette action pour générer une section d'une page HTML, mais pas une page HTML complète. Les vues partielles peuvent être réutilisées dans de nombreuses vues d'une application Web. |
| • | RedirectToRouteResult. Vous pouvez utiliser ce résultat d'action pour rediriger le navigateur Web vers une autre méthode d'action ou une autre route. |
| • | RedirigerRésultat. Vous pouvez utiliser ce résultat d'action pour rediriger vers une URL spécifique, soit dans votre application Web, soit vers un emplacement externe. |
| • | ContentResult. Vous pouvez utiliser ce résultat d'action pour renvoyer du texte au navigateur Web. Vous pouvez renvoyer du texte brut, du XML, un tableau séparé par des virgules ou d'autres formats de texte. Ce texte peut être rendu dans le navigateur Web ou analysé avec du code côté client. |

**Actions de l'enfant**

Lorsqu'une action renvoie une vue complète, MVC envoie une nouvelle page Web complète au navigateur Web pour affichage. Parfois, vous souhaiterez peut-être appeler une action à partir d'une vue, pour renvoyer un élément de contenu à afficher dans une page Web. Une action enfant est une méthode d'action qui peut renvoyer un petit élément de contenu de cette manière. FileContentResult est souvent un bon exemple d'action enfant, car l'image renvoyée fait généralement partie d'une page Web. Les vues partielles prennent également en charge les actions enfants.

Pour déclarer une méthode d'action en tant qu'action enfant, vous pouvez utiliser l'annotation [ChildActionOnly]. Cette annotation garantit que la méthode d'action ne peut être appelée qu'à partir d'une vue à l'aide de l'assistant Html.Action (). En utilisant cette méthode, vous pouvez empêcher un utilisateur d'appeler l'action enfant directement en tapant l'URL correcte dans la barre d'adresses.

**Question:**Quels sont les différents types de retour ActionResult que vous pouvez écrire sous forme de code lors de la création d'un contrôleur?

## **Utilisation des paramètres**

Lorsque les utilisateurs demandent des pages Web, ils spécifient souvent des informations autres que le nom de la page Web elle-même. Par exemple, lorsqu'ils demandent une page de détails sur le produit, ils peuvent spécifier le nom ou le numéro de catalogue du produit à afficher. Ces informations supplémentaires sont appelées paramètres. Vous devez comprendre comment déterminer dans le code les paramètres que l'utilisateur a envoyés dans sa demande.

Les classes ControllerActionInvoker et DefaultModelBinder obtiennent des paramètres à partir d'une demande utilisateur et les transmettent aux méthodes d'action. DefaultModelBinder peut localiser les paramètres dans un formulaire publié, les valeurs de routage, la chaîne de requête ou dans les fichiers publiés. Si le classeur de modèles trouve un paramètre dans la méthode d'action qui correspond au nom et au type d'un paramètre de la demande, la méthode d'action est appelée et le paramètre est passé à partir de la demande. Cette disposition vous permet d'obtenir et d'utiliser des paramètres dans vos actions. Par exemple, si un utilisateur demande l'URL http://www.adventure-works.com/photo/getphotobytitle/?title=myfirstphoto, vous pouvez facilement obtenir des valeurs de titre dans votre méthode d'action.

**Utilisation d'un paramètre de chaîne de requête**

public ActionResult GetPhotoByTitle (titre de la chaîne)

{

var query = from p en contexte.

où p.Title == title

sélectionnez p;

Photo demandéePhoto = (Photo) query.FirstOrDefault ();

if (requiredPhoto! = null)

{

return View ("Détails", demandéPhoto);

}

autre

{

return HttpNotFound ();

}

}

Notez que le code de la méthode d'action utilise le titre du paramètre pour formuler une requête LINQ to Entities. Dans ce cas, la requête recherche une photo avec le titre spécifié. Les paramètres des méthodes d'action sont fréquemment utilisés de cette manière.

**Noter:**L'exemple fonctionne si DefaultModelBinder passe des paramètres. Si vous créez un classeur de modèles personnalisé dans votre application, vous devez vous assurer qu'il transmet correctement les paramètres. Sinon, la méthode d'action ne peut pas accéder aux paramètres spécifiés par l'utilisateur dans la demande.

**Question:**Comment DefaultModelBinder passe-t-il les paramètres?

## **Transmission d'informations aux vues**

Vous pouvez passer un objet de modèle, tel qu'une photo, de la méthode d'action à la vue à l'aide de la méthode d'assistance View (). Il s'agit d'une méthode fréquemment utilisée pour transmettre des informations d'une action de contrôleur à une vue. En effet, cette méthode adhère étroitement au modèle Model-View-Controller, dans lequel chaque vue restitue les propriétés trouvées dans la classe de modèle, que la vue reçoit du contrôleur. Vous devez utiliser cette approche dans la mesure du possible.

Cependant, dans certains cas, vous souhaiterez peut-être augmenter les informations de la classe de modèle avec des valeurs supplémentaires. Par exemple, vous souhaiterez peut-être envoyer un titre, qui doit être inséré dans l'en-tête de page, à la vue. De plus, certaines vues n'utilisent pas de classes de modèle. La page d'accueil d'un site Web, par exemple, n'a souvent pas de classe de modèle spécifique. Pour vous aider dans ces situations, vous pouvez utiliser deux autres méthodes pour fournir des données supplémentaires: ViewBag et ViewData.

**Utilisation du ViewBag**

Le ViewBag est un objet dynamique qui fait partie de la classe de contrôleur de base. Comme il s'agit d'un objet dynamique, vous pouvez lui ajouter des propriétés de n'importe quel type, dans la méthode d'action. Dans la vue, vous pouvez utiliser l'objet ViewBag pour obtenir les valeurs ajoutées dans l'action.

**Ajout de propriétés à l'objet ViewBag**

ViewBag.Message = "Ce texte n'est pas dans l'objet modèle";

ViewBag.ServerTime = DateTime.Now;

**Utilisation des propriétés ViewBag**

<p>

Le message du jour est: @ ViewBag.Message

</p>

<p>

L'heure sur le serveur est: @ ViewBag.ServerTime.ToString ()

</p>

**Utilisation du dictionnaire ViewData**

L'objet ViewBag a été ajouté à MVC dans la version 3. Dans les versions antérieures, vous pouviez transmettre des données supplémentaires aux vues à l'aide du dictionnaire ViewData. Cette fonctionnalité est toujours disponible dans MVC 5 pour la compatibilité descendante et pour les développeurs qui préfèrent utiliser des objets de dictionnaire. En fait, ViewBag est un wrapper dynamique au-dessus du dictionnaire ViewData. Cela signifie que vous pouvez enregistrer une valeur dans une action de contrôleur à l'aide de ViewBag et lire la même valeur à l'aide de ViewData.

**Ajout de données à ViewData**

ViewData ["Message"] = "Ce texte n'est pas dans l'objet modèle"

ViewData ["ServerTime"] = DateTime.Now;

**Utilisation des valeurs ViewData**

<p>

Le message du jour est: @ViewData ["Message"]

</p>

<p>

L'heure sur le serveur est: @ ((DateTime) ViewData ["ServerTime"])

</p>

Dans les exemples, notez que vous pouvez convertir toutes les valeurs ViewData autres que des chaînes.

**Question:**ViewBag et ViewData ont-ils des objectifs différents?

## **Démonstration: comment créer un contrôleur**

Dans cette démonstration, vous verrez comment créer un contrôleur et écrire des actions communes dans le contrôleur.

**Noter:**À la fin de cette démonstration, l'application n'inclura pas de vues. Par conséquent, l'application ne peut pas afficher de pages Web.

### **Étapes de démonstration**

|  |  |
| --- | --- |
| • | Vous trouverez les étapes dans la section «Leçon 1: Écriture de contrôleurs et d'actions» sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD04_DEMO.md>. |

## **Que sont les usines de contrôleurs?**

Une fabrique de contrôleurs est un composant MVC qui instancie les classes de contrôleurs que vous créez. Par exemple, lorsqu'un utilisateur demande une liste d'objets de modèle Photo, une fabrique de contrôleurs doit créer une instance de la classe PhotoController. Un appelant d'action appelle ensuite l'une des méthodes d'action de cette classe et un classeur de modèle lui transmet des paramètres.

Le framework MVC comprend une classe DefaultControllerFactory intégrée qui convient à la plupart des applications Web. Cependant, vous devez comprendre comment DefaultControllerFactory détermine la classe de contrôleur qu'il doit créer. Parfois, vous devrez peut-être créer une fabrique de contrôleurs personnalisée pour implémenter votre propre logique de création de contrôleurs.

**Comment la classe DefaultControllerFactory localise une classe de contrôleur**

La classe DefaultControllerFactory identifie les classes de contrôleur à l'aide des critères suivants:

|  |  |
| --- | --- |
| • | La portée de la classe doit être publique. |
| • | La classe ne doit pas être marquée comme abstraite. |
| • | La classe ne doit pas accepter de paramètres génériques. |
| • | La classe doit avoir un nom qui se termine par Controller. |
| • | La classe doit implémenter l'interface IController. |

Lorsque l'application Web MVC démarre, DefaultControllerFactory crée une liste de toutes les classes de l'application qui satisfont à ces critères. Cette liste permet de créer rapidement le bon contrôleur. Pour écrire un contrôleur, vous devez vous assurer que tous les critères mentionnés ci-dessus sont mis en œuvre. Habituellement, vous répondez au critère d'interface IController en héritant de la classe Controller de base.

Par défaut, DefaultControllerFactory oblige toutes les classes de contrôleur à se terminer par le mot Controller. Par exemple, en suivant cette convention, pour la classe de modèle Photo, vous créeriez un contrôleur appelé PhotoController.

**Création d'une fabrique de contrôleurs personnalisée**

Parfois, vous souhaiterez peut-être implémenter une fabrique de contrôleurs personnalisée. Il y a deux raisons courantes de faire cela:

|  |  |
| --- | --- |
| • | Pour modifier les critères de sélection des contrôleurs. Les critères décrits précédemment conviennent à la plupart des applications Web, mais parfois, vous souhaiterez peut-être les modifier. Par exemple, vous ne souhaiterez peut-être pas nommer les contrôleurs avec Controller à la fin, ou vous souhaiterez peut-être ajouter vos propres critères supplémentaires. |
| • | Pour prendre en charge l'injection directe pour les tests. L'injection directe est une technique de programmation qui vous permet de spécifier des classes au moment de l'exécution, au lieu de spécifier des classes lors de l'écriture de code. Ceci est utile pour les tests unitaires, car vous pouvez injecter une classe de test avec des données simulées, au lieu de données réelles. La classe DefaultControllerFactory ne prend pas en charge l'injection directe. |

**Une usine de contrôleurs personnalisés**

classe publique AdWorksControllerFactory: IControllerFactory

{

public IController CreateController (RequestContext requestContext, string ControllerName)

{

Tapez targetType = null;

if (ControllerName == "Photo")

{

targetType = typeof (PhotoController);

}

autre

{

targetType = typeof (GeneralPurposeController);

}

renvoyer targetType == null? null: (IController) Activator.CreateInstance (targetType);

}

public SessionStateBehavior GetControllerSessionBehavior (RequestContext requestContext,

string controllerName)

{

return SessionStateBehavior.Default;

}

public void ReleaseController (contrôleur IController)

{

IDisposable jetable = contrôleur comme IDisposable;

if (jetable! = null)

{

jetable.Dispose ();

}

}

}

Vous devez implémenter les méthodes CreateController, GetControllerSessionBehavior et ReleaseController pour toute fabrique de contrôleurs personnalisée que vous créez.

Par exemple, si le nom du contrôleur transmis à une fabrique de contrôleurs est «Photo», le PhotoController est utilisé. Sinon, le GeneralPurposeController est utilisé. La logique d'un véritable contrôleur personnalisé doit être plus sophistiquée que dans cet exemple. Cependant, l'exemple illustre le code minimal requis pour créer une fabrique de contrôleurs personnalisée.

**Enregistrement d'une fabrique de contrôleurs personnalisés**

Même si vous créez une fabrique de contrôleurs personnalisée dans votre application, MVC utilisera toujours la classe DefaultControllerFactory, sauf si vous enregistrez votre fabrique personnalisée.

**Enregistrement d'une fabrique de contrôleurs personnalisés**

protected void Application\_Start ()

{

ControllerBuilder.Current.SetControllerFactory (nouveau AdWorksControllerFactory ());

}

**Question:**Pouvez-vous créer un contrôleur qui ne se termine pas par «Contrôleur»?

# Leçon 2: Écriture de filtres d'action

Dans certaines situations, vous devrez peut-être exécuter du code avant ou après l'exécution des actions du contrôleur. Avant qu'un utilisateur n'exécute une action qui modifie des données, vous souhaiterez peut-être exécuter le code qui vérifie les détails du compte d'utilisateur. Si vous ajoutez un tel code aux actions elles-mêmes, vous devrez dupliquer le code dans toutes les actions où vous souhaitez que le code s'exécute. Les filtres d'action constituent un moyen pratique d'éviter la duplication de code. Vous devez savoir comment créer et utiliser des filtres d'action dans votre application Web et quand les utiliser.

## **Objectifs de la leçon**

Après avoir terminé cette leçon, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Décrivez les filtres d'action. |
| • | Créez des filtres d'action. |
| • | Déterminez quand utiliser les filtres d'action. |

## **Que sont les filtres?**

Le modèle de programmation MVC impose la séparation des préoccupations. Par exemple, la logique métier dans les classes de modèle est distincte de la logique d'entrée dans les automates et de la logique de l'interface utilisateur dans les vues. Chaque classe de modèle est également clairement distincte des autres classes de modèle. Cependant, il existe des scénarios où les exigences peuvent être pertinentes pour de nombreuses parties de votre application et transcender les limites logiques. Par exemple, l'autorisation doit être effectuée pour de nombreuses actions et contrôleurs sensibles, quels que soient le modèle et les vues qu'ils renvoient. Ces types d'exigences sont connus sous le nom de préoccupations transversales. Certains exemples courants de problèmes transversaux incluent l'autorisation, la journalisation et la mise en cache.

**Filtres**

Les filtres sont des classes MVC que vous pouvez utiliser pour gérer les problèmes transversaux dans votre application Web. Vous pouvez appliquer un filtre à une action de contrôleur en annotant la méthode d'action avec l'attribut approprié. Par exemple, une action annotée avec l'attribut [Authorize] ne peut être exécutée que par des utilisateurs authentifiés. Vous pouvez également appliquer un filtre à chaque action dans un contrôleur en annotant la classe de contrôleur avec l'attribut.

**Types de filtres**

Il existe plusieurs types de filtres que vous pouvez utiliser dans MVC. Ces filtres s'exécutent à des étapes légèrement différentes du processus de demande.

| **Type de filtre** | **Interface** | **Classe par défaut** | **La description** |
| --- | --- | --- | --- |
| Autorisation | IAuthorizationFilter | AuthorizeAttribute | S'exécute avant tout autre filtre et avant le code dans la méthode d'action. Utilisé pour vérifier les droits d'accès d'un utilisateur pour l'action. |
| action | IActionFilter | ActionFilterAttribute | S'exécute avant et après le code dans la méthode d'action. |
| Résultat | IResultFilter | ActionFilterAttribute | S'exécute avant et après le retour d'un résultat d'une méthode d'action. |
| Exception | IExceptionFilter | HandleErrorAttribute | S'exécute uniquement si la méthode d'action ou un autre filtre lève une exception. Utilisé pour gérer les erreurs. |

**Question**: Quel type de filtre utiliserez-vous pour les actions suivantes?

|  |  |
| --- | --- |
| 1. | Intercepter une erreur |
| 2. | Modifier un résultat |
| 3. | Autoriser les utilisateurs |
| 4. | Inspecter une valeur retournée |

## **Création et utilisation de filtres d'action**

Si vous avez une préoccupation transversale dans votre application Web, vous pouvez l'implémenter en créant un filtre d'action personnalisé ou un filtre de résultat personnalisé. Vous pouvez créer des filtres personnalisés en implémentant l'interface IActionFilter ou l'interface IResultFilter. Cependant, la classe de base ActionFilterAttribute implémente les interfaces IActionFilter et IResultFilter pour vous. En dérivant votre filtre de la classe ActionFilterAttribute, vous pouvez créer un filtre unique qui peut exécuter du code avant et après l'exécution de l'action, et avant et après le renvoi du résultat.

**Un filtre d'action personnalisé simple**

classe publique SimpleActionFilter: ActionFilterAttribute

{

public override void OnActionExecuting (ActionExecutingContext filterContext)

{

Debug.WriteLine ("Cet événement déclenché: OnActionExecuting");

}

public override void OnActionExecuted (ActionExecutedContext filterContext)

{

Debug.WriteLine ("Cet événement déclenché: OnActionExecuted");

}

public override void OnResultExecuting (ResultExecutingContext filterContext)

{

Debug.WriteLine ("Cet événement déclenché: OnResultExecuting");

}

public override void OnResultExecuted (ResultExecutedContext filterContext)

{

Debug.WriteLine ("Cet événement déclenché: OnResultExecuted");

}

}

**Noter:**Vous pouvez également créer un filtre d'autorisation personnalisé en implémentant l'interface IAuthorizationFilter. Cependant, l'implémentation par défaut d'AutorizeAttribute est très utile et satisfait presque toutes les exigences d'authentification. Vous devez être prudent lorsque vous remplacez le code de sécurité par défaut dans MVC ou tout autre modèle de programmation. Si vous le faites sans une compréhension complète des implications, vous pouvez introduire des vulnérabilités de sécurité qu'un utilisateur malveillant peut exploiter.

**Utilisation d'un filtre d'action personnalisé**

Une fois que vous avez créé un filtre d'action personnalisé, vous pouvez l'appliquer à n'importe quelle méthode ou classe d'action de votre application Web en annotant la méthode ou la classe avec le nom du filtre d'action.

**Utilisation d'un filtre d'action personnalisé**

public class PhotoController: Controller

{

ContextDB contextDB = nouveau ContextDB ();

[SimpleActionFilter]

Index ActionResult public ()

{

return View ("Index", contextDB.Photos.ToList ());

}

}

**Question:**Quels sont les avantages des filtres d'action personnalisés?

## **Discussion: Scénarios de filtre d'action**

Considérez les scénarios suivants. Dans chaque cas, discutez avec le reste de la classe pour savoir si le scénario nécessite un filtre personnalisé, s'il peut être résolu avec un type de filtre intégré ou s'il ne peut pas être résolu avec des filtres.

|  |  |
| --- | --- |
| 1. | Vous écrivez une application de partage de photos et vous souhaitez permettre à chaque utilisateur de discuter de photos, d'appareils photo, d'objectifs et d'autres équipements photographiques avec d'autres utilisateurs qu'il a marqués comme ses amis. Les autres utilisateurs doivent être empêchés de voir ces discussions. |
| 2. | Vous voulez vous assurer que lorsque MVC appelle la méthode d'action GetImage, l'ID dans la chaîne de requête est passé en tant que paramètre. |
| 3. | Vous rédigez une application de partage de photos et vous souhaitez empêcher les utilisateurs non authentifiés d'ajouter des commentaires à une photo. |
| 4. | Vous souhaitez empêcher les utilisateurs malveillants d'intercepter les informations d'identification saisies par les utilisateurs dans le formulaire de connexion de votre application Web. Vous voulez vous assurer que les informations d'identification sont chiffrées. |

# **Atelier: Développement de contrôleurs ASP.NET MVC 5**

### **Scénario**

Vous avez été invité à ajouter un contrôleur à l'application de partage de photos qui correspond à la classe de modèle Photo que vous avez créée dans un module précédent. Le contrôleur doit inclure des actions qui répondent lorsque les utilisateurs téléchargent des photos, répertorient toutes les photos, affichent une seule photo et suppriment des photos de l'application. Vous devez également ajouter une action qui renvoie la photo sous forme de fichier .jpg à afficher sur une page Web.

Les membres de votre équipe de développement sont nouveaux dans ASP.NET MVC et ils trouvent l'utilisation des actions de contrôleur déroutante. Par conséquent, vous devez les aider en ajoutant un composant qui affiche les paramètres d'action dans la fenêtre Sortie de Visual Studio chaque fois qu'une action s'exécute. Vous allez ajouter un filtre d'action pour y parvenir.

**Noter:**Les contrôleurs et vues que vous avez ajoutés dans le laboratoire du module 3 devaient tester vos nouvelles classes de modèles. Ils ont été supprimés du projet pour créer les contrôleurs réels. Vous allez créer des vues temporaires pour tester ces contrôleurs à la fin de ce TP.

### **Objectifs**

Après avoir terminé cet atelier, vous serez en mesure de:

|  |  |
| --- | --- |
| • | Ajoutez un contrôleur MVC à une application Web. |
| • | Écrivez des actions dans un contrôleur MVC qui répondent aux opérations de l'utilisateur telles que la création, l'indexation, l'affichage et la suppression. |
| • | Écrivez des filtres d'action qui exécutent du code pour plusieurs actions. |

##### **Configuration du laboratoire**

Durée estimée: 60 minutes

Vous trouverez les étapes de haut niveau sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD04_LAB_MANUAL.md>.

Vous trouverez les étapes détaillées sur la page suivante:<https://github.com/MicrosoftLearning/20486-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486C/20486C_MOD04_LAK.md>.

### **Exercice 1: Ajout d'un contrôleur MVC et écriture des actions**

##### **Scénario**

Dans cet exercice, vous allez créer le contrôleur MVC qui gère les opérations photo. Vous ajouterez également les actions suivantes:

|  |  |
| --- | --- |
| • | Indice. Cette action obtient une liste de tous les objets Photo et la transmet à la vue Index pour affichage. |
| • | Affichage. Cette action prend un identifiant pour trouver un seul objet Photo. Il transmet la photo à la vue d'affichage pour affichage. |
| • | Créer (GET). Cette action crée un nouvel objet Photo et le transmet à la vue Créer, qui affiche un formulaire que le visiteur peut utiliser pour télécharger une photo et la décrire. |
| • | Créer (PUBLIER ). Cette action reçoit un objet Photo de la vue Créer et enregistre les détails dans la base de données. |
| • | Supprimer (GET). Cette action affiche un objet Photo et demande à l'utilisateur de confirmer la suppression de l'objet Photo. |
| • | EffacerConfirmé (POST). Cette action supprime un objet Photo après confirmation. |
| • | GetImage:Cette action renvoie l'image photo de la base de données sous forme de fichier JPEG. Cette méthode est appelée par plusieurs vues pour afficher l'image. |

### **Exercice 2: Facultatif - Écriture des filtres d'action dans un contrôleur**

##### **Scénario**

Votre équipe de développement est nouvelle dans MVC et a du mal à transmettre les bons paramètres aux contrôleurs et aux actions. Vous devez implémenter un composant qui affiche les noms de contrôleurs, les noms d'actions, les noms de paramètres et les valeurs dans la fenêtre Sortie de Visual Studio pour résoudre ce problème. Dans cet exercice, vous allez créer un filtre d'action à cet effet.

Effectuez cet exercice si le temps le permet.

### **Exercice 3: Utilisation du contrôleur photo**

##### **Scénario**

Dans cet exercice, vous allez:

|  |  |
| --- | --- |
| • | Créez un index temporaire et affichez des vues à l'aide du code d'échafaudage intégré au modèle d'application Visual Studio MVC. |
| • | Utilisez les vues pour tester les contrôleurs, les actions et les filtres d'action, et exécutez l'application de partage de photos. |

### **Question (s) de révision**

**Vérifiez vos connaissances**

**Découverte**

**Que se passe-t-il si vous cliquez sur les liens Modifier ou Supprimer dans la vue Index du laboratoire?**

Afficher la solution Réinitialiser

**Vérifiez vos connaissances**

**Découverte**

**Pourquoi avez-vous utilisé leActionNameannotation pour leSupprimerConfirméaction dans lePhotoControllerclasser?**

Afficher la solution Réinitialiser

**Vérifiez vos connaissances**

**Découverte**

**Dans le laboratoire, vous avez ajouté deux actions avec le nom,Créer. Pourquoi est-il possible d'ajouter ces actions sans utiliser l'annotation ActionName?**

Afficher la solution Réinitialiser

# **Examen du module et points à retenir**

Dans ce module, vous avez vu le rôle pivot que jouent les contrôleurs et les actions dans la construction d'une application Web MVC. Les contrôleurs et les actions garantissent que l'application répond correctement à chaque demande de l'utilisateur, créent des instances de classes de modèle et transmettent la classe de modèle à une vue pour affichage. Dans le module suivant, vous verrez comment créer et coder des vues pour implémenter l'interface utilisateur pour votre application.

**Meilleur entrainement:**Sauf si vous avez une bonne raison de ne pas le faire, respectez la convention selon laquelle chaque contrôleur doit être nommé pour correspondre à la classe de modèle correspondante, avec «Controller» ajouté. Par exemple, pour la classe de modèle Photo, le contrôleur doit être appelé PhotoController. En conservant cette convention, vous créez un ensemble de classes de modèles et de contrôleurs nommé logiquement et vous pouvez utiliser la fabrique de contrôleurs par défaut.

**Meilleur entrainement:**Faites très attention si vous choisissez de remplacer le filtre AuthorizeAttribute intégré, car il implémente les autorisations et la sécurité de l'application Web. Si vous modifiez négligemment l'infrastructure de sécurité du framework MVC, vous pouvez introduire des vulnérabilités dans votre application. Utilisez plutôt le filtre intégré dans la mesure du possible.

### **Question (s) de révision**