Module04

Developing ASP.NET MVC 4 Controllers

Module Overview

- Writing Controllers and Actions
- Writing Action Filters

Lesson 1: Writing Controllers and Actions

- Responding to User Requests
- Writing Controller Actions
- Using Parameters
- Passing Information to Views
- Demonstration: How to Create a Controller
- What Are Controller Factories?

Responding to User Requests

When an MVC web application receives a user request, the following events occur:

MvcHandler creates a controller factory.

Controller factory creates a **Controller** object and **MvcHandler** calls the **Execute** method.

ControllerActionInvoker examines **RequestContext** and determines the action to call.

ControllerActionInvoker determines the values to be passed to the action as parameters.

ControllerActionInvoker runs the action.

Writing Controller Actions

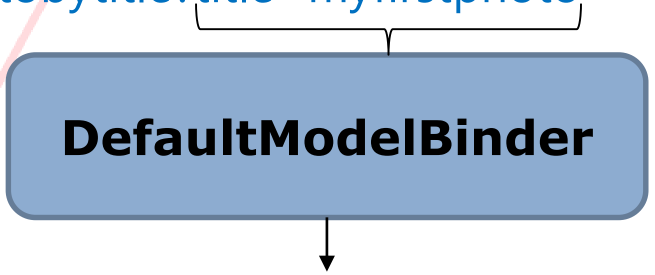
- Writing a Controller action includes:
  - Creating a Simple Details Action
  - Using GET and POST HTTP Verbs in Actions
  - Creating Action Result Classes
  - Creating Child Actions
- Sample controller action

```
public ActionResult First () {
   Photo firstPhoto = context.Photos.ToList()[0];
   if (firstPhoto != null)   {
      return View("Details", firstPhoto);
   } else {
      return HttpNotFound();
   }
}
```

Using Parameters

The **DefaultModelBinder** obtains the **Title** parameter from the query string and passes it to the title parameter of the **GetPhotoByTitle** method, because the names match.

http://www.adventureworks.com/photo/getphotobytitle?title=myfirstphoto



```
public ActionResult GetPhotoByTitle (string title){
  var query = from p in context.Photos
      where p.Title == title
      select p;
  Photo requestedPhoto = (Photo)query.FirstOrDefault();
  return View("Details", requestedPhoto);
}
```

Passing Information to Views

- To pass information to views that have model classes, you can use the:
  - **View() helper** method: To pass information from a controller action to a view
- To pass information to views that do not have model classes, you can use the:
  - **ViewBag** property : To dynamically add objects of any type
  - **ViewData Dictionary** property: Used in MVC 2 to add extra data to views. Available in MVC 4 for backward compatibility

Using the ViewBag Object

Adding Information:

```
ViewBag.Message = "This text is not in the model object";
ViewBag.ServerTime = DateTime.Now;
```

Retrieving Information:

```
<p>
  The message of the day is: @ViewBag.Message
</p>
<p>
  The time on the server is: @ViewBag.ServerTime.ToString()
</p>
```

Demonstration: How to Create a Controller

In this demonstration, you will see how to:

1. Add a controller to an MVC application.
2. Write an action that displays a list of model objects.
3. Write an action that displays the details of a model object.
4. Write HTTP GET and POST actions that create and save a new model object.

What Are Controller Factories?

- Controller factories instantiate the controllers that you create
- You can create a controller factory by:
  - Using the built-in **DefaultControllerFactory** class
  - Creating a custom controller factory for modifying the criteria for selecting controllers or for providing support for testing
    - You need to register custom controller facory by using the **ControllerBuilder** class in the **Global.asax** file

A Custom Controller Factory

```
public class AdWorksControllerFactory : IControllerFactory
{
  public IController CreateController (RequestContext requestContext,
    string ControllerName)
  {
    Type targetType = null;
    if (ControllerName == "Photo") {
      targetType = typeof(PhotoController);
    } else {
      targetType = typeof(GeneralPurposeController);
    }
    return targetType == null ? null :
      (IController)Activator.CreateInstance(targetType);
  }
}
```

Lesson 2: Writing Action Filters

- What Are Filters?
- Creating and Using Action Filters
- Discussion: Action Filter Scenarios

What Are Filters?

- Some requirements cut across logical boundaries are called cross-cutting concerns. Examples include:
  - Authorization
  - Logging
  - Caching
- There are four different types of filters:
  - Authorization filters run before any other filter and before the code in the action method
  - Action filters run before and after the code in the action method
  - Result filters run before and after a result is returned from an action method
  - Exception filters run only if the action method or another filter throws an exception

Creating and Using Action Filters

Sample Action Filter

```
public class SimpleActionFilter : ActionFilterAttribute {
  public override void OnActionExecuting
    (ActionExecutingContext filterContext) {
    Debug.WriteLine("This Event Fired: OnActionExecuting");
  }
  public override void OnActionExecuted
    (ActionExecutedContext filterContext) {
    Debug.WriteLine("This Event Fired: OnActionExecuted");
  }
}
```

Discussion: Action Filter Scenarios

Discuss the following scenarios:

- A photo sharing application with discussion amongst friends
- Passing correct parameters to the **GetImage** method
- Preventing unauthenticated users from adding comments to a photo
- Preventing malicious users from intercepting credentials

Lab: Developing ASP.NET MVC 4 Controllers

- Exercise 1: Adding an MVC Controller and Writing the Actions
- Exercise 2: Optional—Writing the Action Filters in a Controller
- Exercise 3: Using the Photo Controller

Estimated Time: 60 minutes

Lab Scenario

You have been asked to add a controller to the photo sharing application that corresponds to the Photo model class that you have created in an earlier module. The controller should include actions that respond when users upload photos, list all photos, display a single photo, and delete photos from the application. You should also add an action that returns the photo as a .jpg file to show on a webpage.

The members of your development team are new to ASP.NET MVC and they find the use of controller actions confusing. Therefore, you need to help them by adding a component that displays action parameters in the Visual Studio Output window whenever an action runs. You will add an action filter to achieve this.

Note: The controllers and views that you have added in Lab 2 were to test your new model classes. They have been removed from the project to create the actual controllers. You will create temporary views to test these controllers at the end of this lab.

Lab Review

- What will happen if you click the Edit or Delete links in the Index view in the Lab?
- Why did you use the ActionName annotation for the DeleteConfirmed action in the PhotoController class?
- In the lab, you added two actions with the name, Create. Why is it possible to add these actions without using the ActionName annotation?

Module Review and Takeaways

- Review Question(s)
- Best Practice