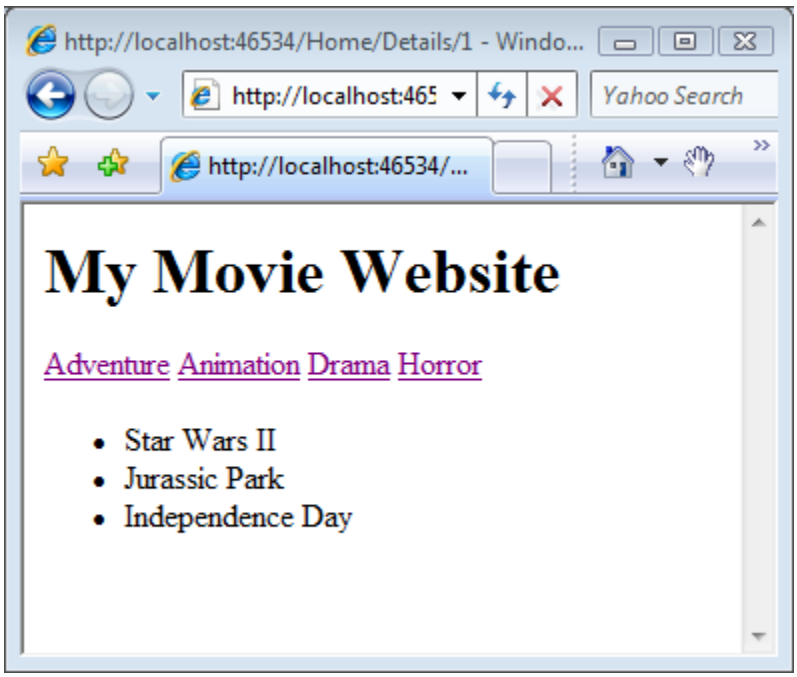
MVC :: Passing Data to View Master Pages

The goal of this tutorial is to explain how you can pass data from a controller to a view master page. We examine two strategies for passing data to a view master page. First, we discuss an easy solution that results in an application that is difficult to maintain. Next, we examine a much better solution that requires a little more initial work but results in a much more maintainable application.

The Problem

Imagine that you are building a movie database application and you want to display the list of movie categories on every page in your application (see Figure 1). Imagine, furthermore, that the list of movie categories is stored in a database table. In that case, it would make sense to retrieve the categories from the database and render the list of movie categories within a view master page.

Figure 1 – Displaying movie categories in a view master page



Here's the problem. How do you retrieve the list of movie categories in the master page? It is tempting to call methods of your model classes in the master page directly. In other words, it is tempting to include the code for retrieving the data from the database right in your master page. However, bypassing your MVC controllers to access the database would violate the clean separation of concerns that is one of the primary benefits of building an MVC application.

In an MVC application, you want all interaction between your MVC views and your MVC model to be handled by your MVC controllers. This separation of concerns results in a more maintainable, adaptable, and testable application.

In an MVC application, all data passed to a view – including a view master page – should be passed to a view by a controller action. Furthermore, the data should be passed by taking

advantage of view data. In the remainder of this tutorial, I examine two methods of passing view data to a view master page.

The Simple Solution

Let's start with the simplest solution to passing view data from a controller to a view master page. The simplest solution is to pass the view data for the master page in each and every controller action.

Consider the controller in Listing 1. It exposes two actions named Index() and Details(). The Index() action method returns every movie in the Movies database table. The Details() action method returns every movie in a particular movie category.

Listing 1 – Controllers\HomeController.cs

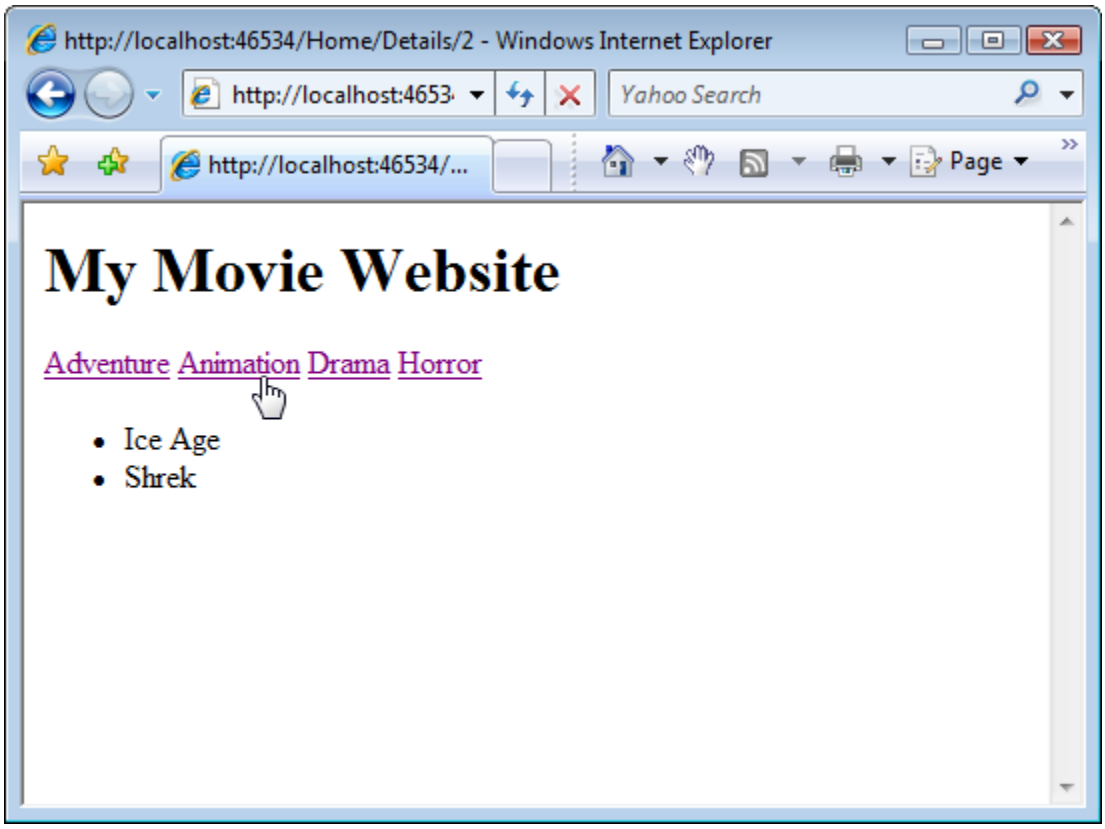
```
using System.Linq;
using System.Web.Mvc;
using MvcApplication1.Models;


namespace MvcApplication1.Controllers
{
    [HandleError]
    public class HomeController : Controller
    {
        private MovieDataContext _dataContext = new
   MovieDataContext();


        /// <summary>
        /// Show list of all movies
        /// </summary>
        public ActionResult Index()
        {
            ViewData["categories"] = from c in
   _dataContext.MovieCategories
                                     select c;
            ViewData["movies"] = from m in _dataContext.Movies
                                 select m;
            return View();
        }


        /// <summary>
```

```
        /// Show list of movies in a category
        /// </summary>
        public ActionResult Details(int id)
        {
            ViewData["categories"] = from c in
   _dataContext.MovieCategories
                                     select c;
            ViewData["movies"] = from m in _dataContext.Movies
                                 where m.CategoryId == id
                                 select m;
            return View();
        }
    }
}
```

Notice that both the Index() and the Details() actions add two items to view data. The Index() action adds two keys: categories and movies. The categories key represents the list of movie categories displayed by the view master page. The movies key represents the list of movies displayed by the Index view page.

The Details() action also adds two keys named categories and movies. The categories key, once again, represents the list of movie categories displayed by the view master page. The movies key represents the list of movies in a particular category displayed by the Details view page (see Figure 2).

Figure 2 – The Details view



The Index view is contained in Listing 2. It simply iterates through the list of movies represented by the movies item in view data.

Listing 2 – Views\Home\Index.aspx

```
<%@ Page Title="" Language="C#"
   MasterPageFile="~/Views/Shared/Site.Master"
   AutoEventWireup="true" CodeBehind="Index.aspx.cs"
   Inherits="MvcApplication1.Views.Home.Index" %>

<%@ Import Namespace="MvcApplication1.Models" %>

<asp:Content ID="Content1"
   ContentPlaceHolderID="ContentPlaceHolder1" runat="server">


<ul>

<% foreach (var m in (IEnumerable<Movie>)ViewData["movies"])

   { %>


    <li><%= m.Title %></li>


<% } %>

</ul>
```

```
</asp:Content>
```

The view master page is contained in Listing 3. The view master page iterates and renders all of the movie categories represented by the categories item from view data.

Listing 3 – Views\Shared\Site.master

```
<%@ Master Language="C#" AutoEventWireup="true"
   CodeBehind="Site.Master.cs"
   Inherits="MvcApplication1.Views.Shared.Site" %>

<%@ Import Namespace="MvcApplication1.Models" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">


<html xmlns="http://www.w3.org/1999/xhtml" >

<head runat="server">

    <title></title>

    <asp:ContentPlaceHolder ID="head" runat="server">

    </asp:ContentPlaceHolder>

</head>

<body>

    <div>

       <h1>My Movie Website</h1>


       <% foreach (var c in
   (IEnumerable<MovieCategory>)ViewData["categories"])

          {%>


          <%= Html.ActionLink(c.Name, "Details", new {id=c.Id} ) %>


       <% } %>



        <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
   runat="server">


        </asp:ContentPlaceHolder>

    </div>

</body>
```

```
</html>
```

All data is passed to the view and the view master page through view data. That is the correct way to pass data to the master page.

So, what's wrong with this solution? The problem is that this solution violates the DRY (Don'tt Repeat Yourself) principle. Each and every controller action must add the very same list of movie categories to view data. Having duplicate code in your application makes your application much more difficult to maintain, adapt, and modify.

The Good Solution

In this section, we examine an alternative, and better, solution to passing data from a controller action to a view master page. Instead of adding the movie categories for the master page in each and every controller action, we add the movie categories to the view data only once. All view data used by the view master page is added in an Application controller.

The ApplicationController class is contained in Listing 4.

Listing 4 – Controllers\ApplicationController.cs

```
using System.Linq;
using System.Web.Mvc;
using MvcApplication1.Models;


namespace MvcApplication1.Controllers
{
    public abstract class ApplicationController : Controller
    {
        private MovieDataContext _dataContext = new
   MovieDataContext();


        public MovieDataContext DataContext
        {
            get { return _dataContext; }
        }


        public ApplicationController()
        {
            ViewData["categories"] = from c in
   DataContext.MovieCategories
                                     select c;
```

```
        }


    }
}
```

There are three things that you should notice about the Application controller in Listing 4. First, notice that the class inherits from the base System.Web.Mvc.Controller class. The Application controller is a controller class.

Second, notice that the Application controller class is an *abstract* class. An abstract class is a class that must be implemented by a concrete class. Because the Application controller is an abstract class, you cannot not invoke any methods defined in the class directly. If you attempt to invoke the Application class directly then you'll get a Resource Cannot Be Found error message.

Third, notice that the Application controller contains a constructor that adds the list of movie categories to view data. Every controller class that inherits from the Application controller calls the Application controller's constructor automatically. Whenever you call any action on any controller that inherits from the Application controller, the movie categories is included in the view data automatically.

The Movies controller in Listing 5 inherits from the Application controller.

Listing 5 – Controllers\MoviesController.cs

```
using System.Linq;
using System.Web.Mvc;

namespace MvcApplication1.Controllers
{
    public class MoviesController : ApplicationController
    {
        /// <summary>
        /// Show list of all movies
        /// </summary>
        public ActionResult Index()
        {
            ViewData["movies"] = from m in DataContext.Movies
                                 select m;
            return View();
        }

        /// <summary>
```

```
        /// Show list of movies in a category
        /// </summary>
        public ActionResult Details(int id)
        {
            ViewData["movies"] = from m in DataContext.Movies
                                 where m.CategoryId == id
                                 select m;
            return View();
        }

    }
}
```

The Movies controller, just like the Home controller discussed in the previous section, exposes two action methods named Index() and Details(). Notice that the list of movie categories displayed by the view master page is not added to view data in either the Index() or Details() method. Because the Movies controller inherits from the Application controller, the list of movie categories is added to view data automatically.

Notice that this solution to adding view data for a view master page does not violate the DRY (Don't Repeat Yourself) principle. The code for adding the list of movie categories to view data is contained in only one location: the constructor for the Application controller.

Summary

In this tutorial, we discussed two approaches to passing view data from a controller to a view master page. First, we examined a simple, but difficult to maintain approach. In the first section, we discussed how you can add view data for a view master page in each every controller action in your application. We concluded that this was a bad approach because it violates the DRY (Don't Repeat Yourself) principle.

Next, we examined a much better strategy for adding data required by a view master page to view data. Instead of adding the view data in each and every controller action, we added the view data only once within an Application controller. That way, you can avoid duplicate code when passing data to a view master page in an ASP.NET MVC application.