Overview MVC4

FPT Software | FSU1.BU5

Training MVC4

2015

1. **Introduction**

These are what I think is the most basic and easy to understand for MVC technology. I research and read the document from the internet and then filtered again, summarize to easiest and most accessible in below to hope every body can get an overview of MVC and know how to work with it from the simplest projects.

1. **Definition**
2. **Definition MVC:**

**Model–view–controller** (**MVC**) is a software [architectural pattern](http://en.wikipedia.org/wiki/Architectural_pattern) for implementing [user interfaces](http://en.wikipedia.org/wiki/User_interface). It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

The MVC model defines web applications with 3 logic layers: The business layer (Model logic), the display layer (View logic), the input control (Controller logic).

**The Model** is the part of the application that handles the logic for the application data.  
Often model objects retrieve data (and store data) from a database.

**The View** is the parts of the application that handles the display of the data. Most often the views are created from the model data.

**The Controller** is the part of the application that handles user interaction. Typically controllers read data from a view, control user input, and send input data to the model.

The MVC separation helps you manage complex applications, because you can focus on one aspect a time. For example, you can focus on the view without depending on the business logic. It also makes it easier to test an application.

The MVC separation also simplifies group development. Different developers can work on the view, the controller logic, and the business logic in parallel.

1. **Benefits of ASP.NET MVC:**
2. Loser To The Protocol
3. Separation Of Concerns
4. Testability
5. **The Razor view engine**

One of the core components of the new ASP.NET Web Pages technology is the Razor view  
engine. This engine provides a concise way to mix code and markup within the same file.  
ASP.NET MVC applications can also make use of the Razor view engine as an alternative  
to the Web Forms view engine that was available in both ASP.NET MVC 1 and 2.  
As an example, the following code snippet shows a simple page that constructs a  
list of product names using the older Web Forms view engine:

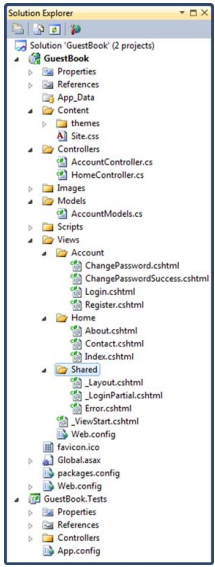
<%@ Page Language="C#" Inherits="System.Web.Mvc.ViewPage<Product[]>" %>  
<ul>  
<% foreach(var product in Model) { %>  
<li><%: product.Name %></li>  
<% } %>  
</ul>

This is quite verbose. The Page declaration at the top and the code nuggets (<% and  
%>) that are used to switch between code and markup add a lot of additional characters to the page markup. By contrast, Razor provides a much cleaner way to achieve  
the same result:

@model Product[]  
<ul>  
@foreach(var product in Model) {  
<li>@product.Name</li>  
}  
</ul>

As you can see, Razor does not require code nuggets to transition between code and  
markup, which helps to keep view logic much more focused on the page’s markup.

1. **A tour of the default project template**

With the newly created project open, you’ll notice that the default project template comes with several subdirectories containing various files. Each of the files and directories within the default project template serves a specific purpose. We’ll take a look at each one in turn:

THE APP\_DATA DIRECTORY  
The App\_Data directory can be used to store databases, XML files, or any other data that your application needs. The ASP.NET runtime understands this special directory and will prevent users from accessing files in it directly. Only your application can read and write to this directory.

THE CONTENT DIRECTORY  
The purpose of the Content directory is to contain any noncode assets that need to be deployed with your application. These typically include images and CSS files (stylesheets). By default, the Content directory contains the default stylesheet used by the project (Site.css) as well as a themes subdirectory that contains images and CSS for use with jQuery UI (which is a client-side framework for user-interface).

THE CONTROLLERS DIRECTORY  
The controller is the coordinator that is responsible for processing input and then deciding which actions should be performed (such as rendering a view). In ASP.NET MVC, controllers are represented as classes within the Controllers directory. By default, this directory contains two controllers - the HomeController (which handles requests for your home page) and the AccountController (which handles authentication).

THE MODELS DIRECTORY  
The Models directory is typically used to contain any classes that represent the core concepts of your application, or classes that hold data in a format that is specific to a particular view (a *view model*). As your applications get larger, you may decide that you wish to move these classes into a separate project, but keeping them in the Models directory is a good starting point for small projects. The default project contains a single file in this directory - AccountModels.cs. It contains several classes related to authentication that are used by the default project template.

THE SCRIPTS DIRECTORY

The Scripts directory is where you can place any JavaScript files used by your application. The default project template contains quite a lot of files in this directory, including the popular open-source jQuery library and scripts used for performing client-side validation.

THE VIEWS DIRECTORY

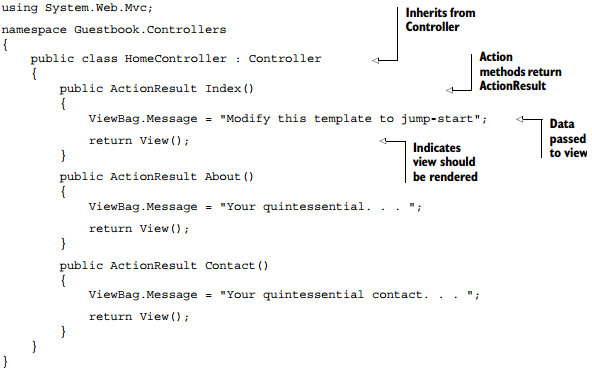
The Views directory contains the templates used to render your user interface. Each of these templates is represented as a Razor view (.cshtml file) within a subdirectory named after the controller responsible for rendering that view.

GLOBAL.ASAX  
The Global.asax file lives in the root of the project structure and contains initialization code that runs when the application is first started up, such as code that registers routes.

WEB.CONFIG  
The Web.config file also lives in the root of the application and contains configuration details necessary for ASP.NET MVC to run correctly. Now that you’ve seen a high-level overview of the different files in the default project template, we’ll explore in more detail how the core concepts of controllers, actions, and views interact with one another. We’ll use the default HomeController to illustrate this before creating some controllers of our own.

1. **Controllers, actions, and displaying dynamic content**

CONTROLLER CLASSES AND ACTION METHODS  
In ASP.NET MVC, controllers are represented as classes that inherit from the Controller  
base class, where individual methods (known as *actions*) correspond to individual URLs.  
To illustrate how this works, we’ll take a look at our project’s HomeController, which can  
be found within the Controllers directory. The code for this class is shown in the following listing.

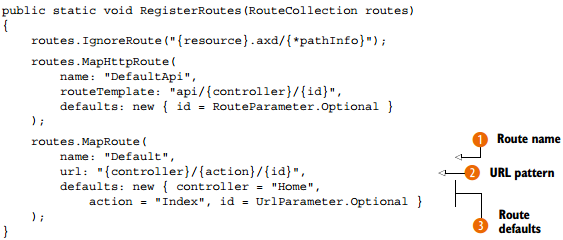


The HomeController is a very straightforward implementation of a controller class. To indicate that it is a controller, it inherits from the Controller base class and also has the “Controller” suffix in its name.

The class also contains two action methods. Actions are public methods on a controller class that handle requests to particular URLs. In this case, the actions are named Index and About. Because these actions are within the HomeController, they can be accessed at the URLs /Home/Index and /Home/About respectively. So if your application were hosted under the domain MySite.com, then the full URL for the Home action would be http://MySite.com/home/index. If a user were to enter this URL into a browser, an instance of the HomeController class would be instantiated by the framework, and the Index action method would be invoked.

ROUTES—MAPPING URLS TO ACTIONS

At this point, you might be asking how does the framework know how to map URLs to a particular controller action? The answer lies within the Global.asax file’s RegisterRoutes method. This method defines routes that map a URL pattern to a controller or action. The implementation of this method is shown next.



The DefaultApi piece is for web API and will be covered in chapter 24. Notice that two entries are defined. The first is an IgnoreRoute, and it basically tells the framework not to worry about anything matching the specified path. In this case, it says not to process any paths containing the .axd file extension, such as Trace.axd. The second entry, MapRoute, defines how URLs are processed. This default route will suffice for a while, but later on you’ll want to add more routes in order to provide URLs that are specific to your application.

Each route has a name 1, a URL definition 2, and optional default values 3. Our first request for / doesn’t have any of these URL pieces, so we’ll look to the default values:

■ controller—"Home"  
■ action—"Index"  
■ id—Optional; allows the id to be omitted from the URL

Listing 2.2 Registering routes  
Because of these default values, you can omit segments from the URL and achieve the same behavior. Again, if your domain were MySite.com, the URLs http://MySite.com/Home/Index, http://MySite.com/Home and http://MySite.com would all end up invoking the HomeController’s Index action.

1. **Entity Framework**

**Entity Framework** (EF) is an open source [object-relational mapping](http://en.wikipedia.org/wiki/Object-relational_mapping) (ORM) framework for [ADO.NET](http://en.wikipedia.org/wiki/ADO.NET), part of [.NET Framework](http://en.wikipedia.org/wiki/.NET_Framework).

The Entity Framework is a set of technologies in ADO.NET that support the development of data-oriented software applications. Architects and developers of data-oriented applications have typically struggled with the need to achieve two very different objectives. They must model the entities, relationships, and logic of the business problems they are solving, and they must also work with the data engines used to store and retrieve the data. The data may span multiple storage systems, each with its own protocols; even applications that work with a single storage system must balance the requirements of the storage system against the requirements of writing efficient and maintainable application code.

The Entity Framework enables developers to work with data in the form of domain-specific objects and properties, such as customers and customer addresses, without having to concern themselves with the underlying database tables and columns where this data is stored. With the Entity Framework, developers can work at a higher level of abstraction when they deal with data, and can create and maintain data-oriented applications with less code than in traditional applications. Because the Entity Framework is a component of the .NET Framework, Entity Framework applications can run on any computer on which the .NET Framework (starting with version 3.5 SP1) is installed.

**Code-First** is a software implementation approach that favors programming (typically imperative, though not necessarily) against an API over other approaches, which may include

**Model-First**. An approach where a model is created using some sort of editor or designer (typically through a highly visual, interactive user interface). Examples include a forms editor for creating user interfaces, a UML modeling tool for generating code or database schemas, or a wizard to generate XML configuration information.

***External Source*** -**First**. A general approach that requires some source of information used to generate program behavior or structure. Either an interactive or automated tool may be used. Examples of external sources include XML schemas, and configuration files.

**Database-First** could be considered a specialization in which a database is used to generate program entities.

1. **Dependency injection**

**Dependency injection** is a [software design pattern](http://en.wikipedia.org/wiki/Software_design_pattern) that implements [inversion of control](http://en.wikipedia.org/wiki/Inversion_of_control) and allows a program design to follow the [dependency inversion principle](http://en.wikipedia.org/wiki/Dependency_inversion_principle). An injection is the passing of a [dependency](http://en.wikipedia.org/wiki/Coupling_(computer_programming)) (a service) to a dependent [object](http://en.wikipedia.org/wiki/Object_(computer_science)) (a client). The service is made part of the client's state. Passing the service to the client, rather than allowing a client to build or find the service, is the fundamental requirement of the pattern.

1. **JqGrid**

jqGrid is an Ajax-enabled JavaScript control that provides solutions for representing and manipulating tabular data on the web. Since the grid is a client-side solution loading data dynamically through Ajax callbacks, it can be integrated with any server-side technology, including PHP, ASP, Java Servlets, JSP, ColdFusion, and Perl. jqGrid uses a jQuery Java Script Library and is written as plugin for that package.

1. **JqPlot**

jqPlot is a plotting and charting plugin for the jQuery Javascript framework. jqPlot produces beautiful line, bar and pie charts with many features:

Numerous chart style options.

Date axes with customizable formatting.

Up to 9 Y axes.

Rotated axis text.

Automatic trend line computation.

Tooltips and data point highlighting.

Sensible defaults for ease of use.

1. **Assignment & Demo**
2. **MVC4, Code first Entity Framework, JqGrid (Refer at TodoList project)**

**Creating simple TodoList application using MVC4, Code first Entity Framework and JqGrid.**

TodoList is a simple web application to create, store and modify Todo tasks to be maintained by the users, which comprises of following fields to the user (Task Name, Task Description, Severity, Target Date, Task Status).

TodoList web application is created using MVC4 architecture, code-first Entity Framework (ORM) and Jqgrid for displaying the data.

**Features of JQGrid**

* JQGrid helps us to develop most browser compatibility web pages and also supports cross browser support functionality.
* CSS based themes. Developers can change the grid skin by defining their own UI CSS framework.
* The new rendering engine is 5-10 times faster loading speed then the previous.
* Pagination functionality is present. So no need to retrieve all the data from the server.
* Sorting, Various data types and sub grid support functionality.
* Event handlers and User API.
* Formatter supports advanced formating of the contents of the cell in the required format in the client side itself.
* Inline Editing: easy to update the cell content in a particular row.
* Searching and filtering.
* Import and Export data.

1. **MVC4, Database first Entity Framework, Razor**
2. **Authentication**