Гайдар Магдануров
Владимир Юнев

ASP.NET MVC Framework

Санкт-Петербург
«БХВ-Петербург»
2010

УДК 681.3.06
ББК 32.973.26-018.2
М12

Магдануров, Г. И.

М12 ASP.NET MVC Framework / Г. И. Магдануров, В. А. Юнев. — СПб.: БХВ-Петербург, 2010. — 320 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-0462-1

Рассмотрены основные принципы и возможности технологии ASP.NET MVC Framework и показаны способы ее практического использования при разработке веб-приложений. Описаны преимущества подхода разработки MVC и рассмотрена структура MVC-приложения. Приведено сравнение технологии WebForms и MVC Framework и рассмотрены вопросы их совмещения.

Описаны модель и доступ к данным (технологии LINQ, Entity Framework и др.), контроллеры, представление и интерфейс приложения, механизмы маршрутизации и Ajax-функциональность. Уделено внимание вопросам тестирования веб-приложений. Рассмотрены особенности применения ASP.NET MVC 2 в Visual Studio 2010.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 30.04.10.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 25,8.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

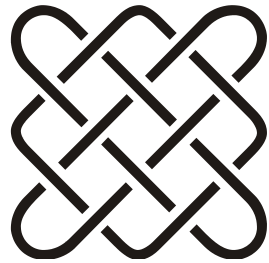
Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.60.953.Д.005770.05.09 от 26.05.2009 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0462-1

Оглавление

ГЛАВА 1



Знакомство с MVC Framework

Первая глава книги посвящена знакомству с подходом разработки веб-приложений на платформе ASP.NET с использованием MVC Framework. Поскольку эта книга рассчитана на разработчиков с разным опытом создания программного обеспечения, прежде чем рассматривать подход к разработке веб-приложений на основе MVC Framework, мы поговорим об основных принципах архитектуры MVC. Прочитав эту главу, вы узнаете об основных компонентах MVC Framework и о том, как эти компоненты находят свое отражение в коде приложения.

Если вы уже знакомы в общих чертах с MVC Framework, можете смело пропустить эту главу и перейти к более детальному изучению составляющих MVC Framework в последующих главах.

Паттерн проектирования MVC

Аббревиатура MVC, давшая название MVC Framework, скрывает в себе всю суть архитектурного подхода построения приложений по принципу MVC: модель, представление и контроллер — это те компоненты, из которых состоит каждое приложение, созданное в этой парадигме.

Приложение, построенное с использованием паттерна проектирования MVC, разбивается на три слабосвязанных между собой логических компонента.

- *Модель* — компонент приложения, отвечающий за взаимодействие с источником данных (база данных, XML-файлы, файловая система и т. п.), а также содержащий описание объектов, описывающих данные, с которыми работает приложение.
- *Представление* — компонент, отвечающий за отображение пользовательского интерфейса — в случае веб-приложения HTML-разметки или других форматов данных, принимаемых вызывающим клиентом.

- *Контроллер* — компонент, содержащий логику приложения. В контроллере описана логика взаимодействия с пользователем — в случае веб-приложения логика обработки HTTP-запросов к веб-приложению. Контроллер взаимодействует с объектами модели, которые, в свою очередь, влияют на представление.

Графическое представление архитектуры MVC приведено на рис. 1.1.

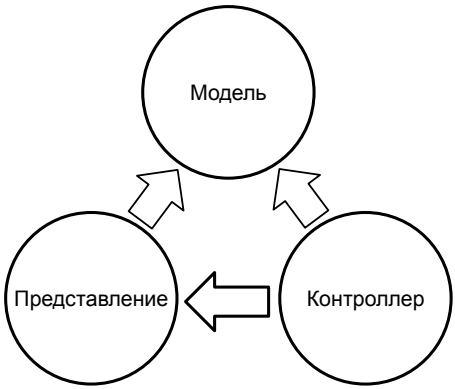


Рис. 1.1. Архитектура MVC

Важно отметить слабую связанность компонентов между собой. Компонент *модель* автономен и не зависит от реализации контроллеров и представлений, его реализация не зависит от реализации остальной части приложений.

С точки зрения контроллера и представления модель представляет собой черный ящик, из которого приходят или в которой помещаются определенные объекты. Говоря простым языком, модель предоставляет контроллеру и представлению некоторый контракт, в соответствии с которым все три компонента системы работают с данными в одинаковом формате.

Примечание

Преимущество такого подхода можно продемонстрировать на примере из жизни. В свое время один из авторов книги работал над Windows-приложением для ведения небольшой складской базы. Это приложение было построено на базе Windows Forms, в полной мере подход MVC не использовался, однако в приложении был выделенный автономный компонент для работы с данными — модель. В дальнейшем, когда потребовалось создать веб-приложение для доступа к складской базе данных из внутренней сети и Интернета, сборка, содержащая компонент *модель*, была подключена к проекту веб-приложения и использовалась совместно с работающим на сервере Windows Forms-приложением. В дальнейшем этой же моделью данных воспользовались веб-службы, которые были разработаны поверх складской базы данных.

Компонент *представление* отображает состояние объектов модели, которое модифицируется контроллером. Представлению недоступны сведения о внутренней реализации контроллера, а также работа представления не зависит от того, каким образом было модифицировано состояние объектов модели. Задача представления очень проста — отображение актуального состояния объектов модели.

Компонент *контроллер* не зависит от внутренней реализации модели и представлений, используемых для визуализации состояния модели, с которой работает контроллер. Задача контроллера заключается в том, чтобы получить запрос от пользователя, обработать его и подготовить коллекцию данных, которые будут использованы представлением.

Примечание

Независимость контроллера и представления позволяет просто заменять представления без необходимости модификации контроллера, что позволяет упростить модификацию системы и получить дополнительное преимущество — использование разных представлений для различных клиентов. На примере из жизни преимущества такого подхода продемонстрировать очень легко. Перед одним из авторов книги как-то встала задача создать версию веб-сайта для мобильных устройств. Поскольку сам проект был разработан с использованием подхода MVC, задачу удалось решить очень просто — в момент обращения к той или иной странице происходила проверка типа браузера пользователя, и в соответствии с этим автоматически подставлялось представление, оптимизированное под этот браузер. При этом не пришлось изменять логику самих контроллеров и URL-адреса.

История паттерна проектирования MVC

Подход к разработке программы с использованием разбиения на компоненты "модель", "представление" и "контроллер" был предложен в 1979 году норвежским ученым Тригве Ринскаугом, когда он работал в исследовательском центре Xerox в Пало Альто. Реализован подход был на языке Smalltalk для графических интерфейсов настольных систем. Тогда веб-приложения еще даже не маячили на горизонте, однако проблема разделения программной логики и интерфейса стояла остро — различные операционные системы предлагали разные подсистемы для отображения графического интерфейса, и при необходимости переноса приложений между платформами возможность повторного использования программной логики при изменении только кода, работающего с графической подсистемой, оказалась весьма кстати.

После того как Интернет прочно вошел в жизнь пользователей по всему миру, появилось большое количество языков, платформ и технологий разработки приложений. Для веб-приложений используются различные подходы раз-

работки, и MVC является одним из наиболее популярных. Убедиться в этом просто, достаточно поискать с помощью любой популярной поисковой системы информацию о библиотеках для реализации подхода MVC.

Примечание

Даже быстрый поиск позволяет получить большое количество результатов: Maverick.NET, Monorail, ProMesh.NET, PureMVC, Mach-II, Model-Glue, FuseBox, Aranea, Cocoon, Grails, GWT, Spring, Struts, Stripes, Tapestry, WebObjects, Wicket, JSF, SproutCore, Wawemaker, Dojo, Catalyst, CGI:Application, Solstice, Gantry, CakePHP, Joomla, Odin Assemble, Prado, Solar, Zand Framework, Symfony, Django, Pylons, Enthought, Zope, web2py, Camping, Merb, Nitro, Ramaze, Ruby on Rails, XForms.

В ноябре 2002 года паттерн MVC был выбран для поддержки в новом стандарте веб-форм XForms, входящем в спецификацию XHTML 2.0, на момент написания этой книги все еще находящейся в статусе черновика.

В декабре 2007 года на суд публике была представлена первая предварительная версия ASP.NET MVC Framework, финальная версия 1.0 которого была выпущена в марте 2009 года.

Преимущества подхода разработки MVC

Использование MVC Framework для веб-приложений на базе ASP.NET приносит ряд преимуществ при разработке и поддержке решений. Выделим три основные преимущества, с которыми разработчик сталкивается, едва преступив к разработке.

1. Полный контроль над кодом разметки

При использовании подходов к разработке веб-приложений, предполагающих автоматическую генерацию кода разметки, таких как WebForms для ASP.NET, разработчик теряет контроль над финальной разметкой, которую получает браузер пользователя. В результате полученные страницы могут не удовлетворять требованиям конечного пользователя — некорректно отображаться в некоторых браузерах или содержать избыточный код разметки.

Полный контроль над разметкой может быть особенно важен, если в веб-приложении активно используется код, работающий на стороне клиента в браузере пользователя.

2. Расширяемость

Компоненты MVC Framework разработаны таким образом, чтобы обеспечивать максимальную расширяемость самой библиотеки. Имеется возможность

использования разных библиотек для обработки представлений, собственных алгоритмов создания объектов контроллеров, а также расширение внутренних механизмов функционирования компонентов библиотеки.

Поскольку все компоненты в MVC Framework могут быть расширены или заменены на собственные, разработчики могут воспользоваться MVC Framework в качестве основы для создания собственной среды разработки веб-приложений, как это часто делают веб-студии, работающие над множеством проектов. В этом случае MVC Framework задает общее направление и стиль разработки веб-приложений, предоставляя разработчику полную свободу выбора уровня абстракции и подходов, используемых в проектах. Можно сказать, что *MVC Framework позволяет сделать разработку веб-приложений настолько простой или настолько сложной, насколько этого хочет сам разработчик*.

Кроме того, стоит особо отметить, что полный исходный код библиотеки MVC Framework доступен публично и распространяется под лицензией, допускающей модификацию исходного кода для собственных нужд.

3. Простота автоматического тестирования

За счет того, что компоненты "модель", "представление" и "контроллер" практически независимы друг от друга, значительно упрощается автоматическое тестирование логики работы веб-приложений.

Если для тестирования программного кода логики, содержащегося в контроллерах, нет необходимости использовать "боевые" данные из реального источника данных, то можно подменить модель версией, созданной специально для тестирования и предоставляющей только минимально необходимый набор данных для целей тестирования и максимальную производительность. При этом вовсе нет необходимости использовать код представлений, что существенно упрощает написание автоматических тестов, поскольку работа идет только с кодом логики, а не с генерируемой браузером разметкой.

Установка MVC Framework

Для того чтобы разрабатывать приложения с использованием MVC Framework, необходимо установить следующие доступные бесплатно для скачивания компоненты:

- среда разработки Visual Web Developer 2008 Express Edition **http://www.microsoft.com/express/download/** (русская версия **http://www.microsoft.com/rus/express/**, приложена на диске);

- сервер баз данных SQL Server 2008 Express Edition **http://www.microsoft.com/express/sql/download/** (русская версия **http://www.microsoft.com/rus/express/sql/download/**, приложена на диске);
- библиотека Microsoft ASP.NET MVC **http://www.asp.net/mvc/download/**.

Установка этих компонентов не должна вызвать сложностей — достаточно запустить мастер установки каждого из компонентов в указанном порядке и следовать инструкциям на экране.

MVC Framework работает с Visual Web Developer Express и со всеми старшими редакциями Visual Studio, если для них установлен компонент Visual Web Developer, поэтому вы можете использовать более старшую редакцию, если она вам доступна. Аналогично можно использовать любую редакцию SQL Server.

Выбор языковой версии не влияет на разработку веб-приложений на MVC Framework, однако в этой книге приводятся снимки экрана английской версии Visual Studio, поскольку на момент написания книги английская версия была более распространена среди русскоговорящих разработчиков.

Первое приложение на MVC Framework

После установки MVC Framework в списке доступных проектов в Visual Studio появится пункт **ASP.NET MVC Web Application**. Этот тип проекта доступен только в случае, если выбрана версия .NET Framework 3.5 (рис. 1.2).

Во время создания проекта, который мы в дальнейшем будем называть *MVC-приложением*, Visual Studio предложит сразу же создать оснастку для тестирования веб-приложения (рис. 1.3). Мы не будем затрагивать тему тестирования MVC-приложений до соответствующей главы, поэтому при создании первого приложения откажемся от создания проекта тестирования.

Visual Studio создаст заготовку проекта MVC-приложения (рис. 1.4), которой мы воспользуемся для изучения составляющих компонентов MVC-приложений.

Проект MVC-приложения представляет собой типовой проект ASP.NET Web Application (Веб-приложение ASP.NET). Напомним, что в отличие от проекта веб-сайта ASP.NET (ASP.NET Web Site), когда проект описывается хранящимися в директории проекта файлами, веб-приложение содержит еще и файл с описанием проекта, в котором описаны основные настройки компиляции и отладки проекта, а также указаны пути к файлам, включенным в проект.

Особенно следует отметить, что MVC-приложения используют стандартную инфраструктуру ASP.NET и, фактически, являются обычными ASP.NET-приложениями, к которым подключены компоненты HttpHandler, меняющие

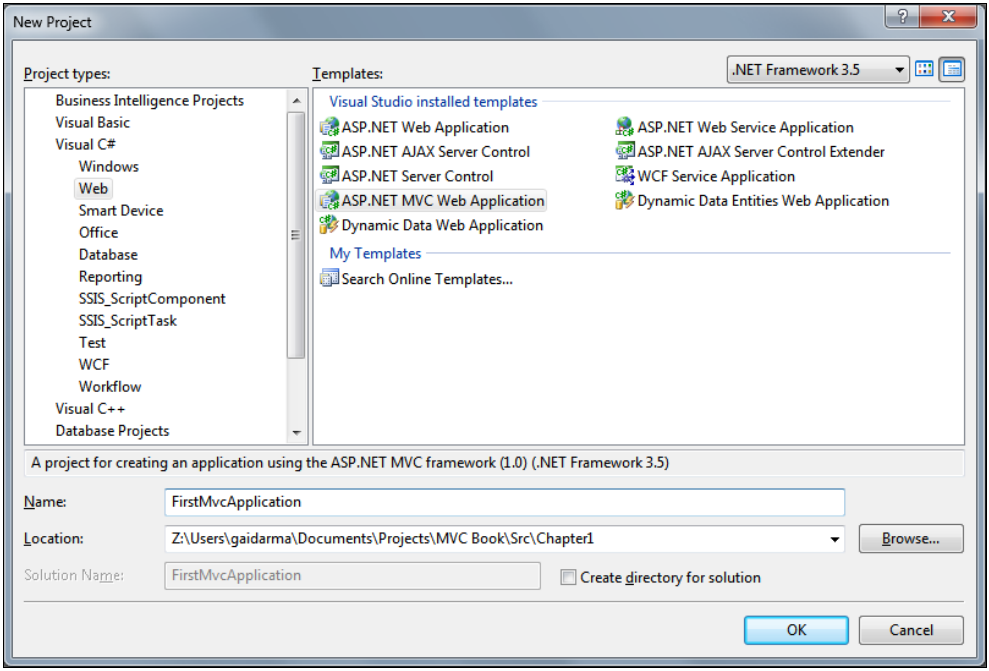


Рис. 1.2. Тип проекта **ASP.NET MVC Web Application** в окне создания новых проектов Visual Studio 2008

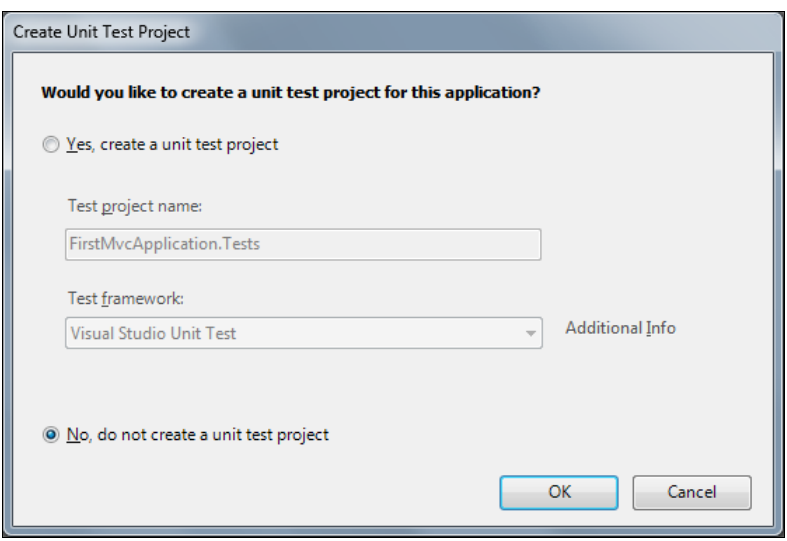


Рис. 1.3. Окно мастера создания проекта для тестирования MVC-приложения

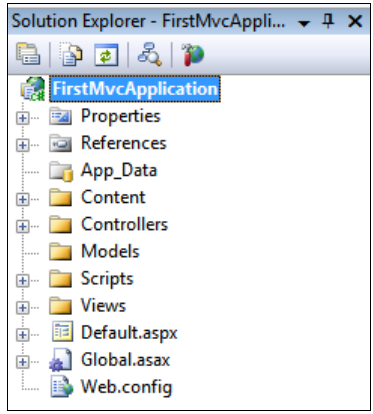


Рис. 1.4. Заготовка MVC-приложения, созданного Visual Studio

логику обработки запросов с целью использования парадигмы MVC. В то же время, в приложении используются файлы Web.config для хранения конфигурации всего веб-приложения и Global.asax для создания глобальных обработчиков событий уровня всего приложения, точно так же, как и в классических ASP.NET-приложениях, использующих компоненты WebForms.

Структура MVC-приложения

Рассмотрим то, как представлены основные компоненты MVC-приложения в виде физических файлов на примере созданного Visual Studio проекта-заготовки.

В структуре папок MVC-приложения важны только две папки: Controlles, в которой размещается код контроллеров, и Views, в которой размещается код представлений. Все остальные папки могут быть произвольно переименованы или удалены из проекта.

Папка Content

В папке Content предлагается размещать файлы, используемые для создания интерфейса приложения на стороне клиента и загружаемые с сервера без изменений. В проекте-заготовке в папке Content размещен файл Site.css, представляющий собой каскадную таблицу стилей для страниц проекта.

Папка Controllers

В папке Controllers размещаются файлы с логикой контроллеров (в нашем случае, поскольку был выбран язык C#, файлы имеют расширение cs). Каждый файл отвечает классу контроллера.

Применяется следующее именование классов контроллеров, которое в свою очередь используется для именования файлов: `ИмяКонтроллераController`.

В проекте-заготовке созданы два файла контроллера: `HomeController`, отвечающий за логику страничек сайта, и `AccountController`, отвечающий за логику регистрации и авторизации пользователей.

Чуть далее в этой главе описана внутренняя структура классов контроллеров.

Папка Models

В папке Models предлагается размещать файлы с логикой взаимодействия с базой данных (модель в паттерне MVC). В приложении-заготовке не генерируется код для работы с данными, поскольку это приложение очень простое и лишь предоставляет базовую структуру. В связи с этим созданная Visual Studio папка Models пуста.

Папка Scripts

В папке Scripts предлагается размещать файлы с кодом клиентских скриптов, используемых на клиенте. В проекте-заготовке уже включены библиотеки JavaScript-кода: MicrosoftAjax — содержащая код для создания клиентской инфраструктуры Microsoft Ajax, MicrosoftAjaxMvc — содержащая код, работающий поверх инфраструктуры Microsoft Ajax и используемый MVC Framework для реализации поддержки асинхронного обновления форм, а также jQuery — популярная библиотека, предоставляющая методы для манипуляции с объектами HTML-документов.

Подробнее об использовании возможностей клиентских библиотек можно узнать в главе, посвященной клиентскому программированию с использованием MVC Framework.

Папка Views

В папке Views размещаются файлы представления. В качестве стандартного механизма представлений в MVC Framework используются ASPX-файлы для создания представлений, MASTER-файлы для создания шаблонов общей разметки части представлений, а также ASCX-файлы для создания *частичных представлений* для многократного использования в составе других представлений.

Если вы знакомы с WebForms в ASP.NET, то представления по сути являются страницами ASP.NET без использования серверных элементов управления и серверных форм.

В папке Views файлы представлений размещаются следующим образом: для каждого контроллера в папке Views создается вложенная папка по имени это-

го контроллера (для `HomeController` папка Home, для `AccountController` папка Account). Помимо этого используется вложенная папка Shared для представлений, шаблонов и частичных представлений, используемых несколькими представлениями (структура для приложения-заготовки показана на рис. 1.5).

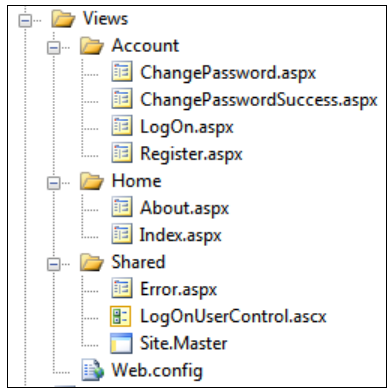


Рис. 1.5. Структура папок представлений для приложения-заготовки

При создании проекта-заготовки в папку Views помещается файл Web.config, описывающий конфигурацию этой папки. В этом файле для папки назначен специальный обработчик, запрещающий доступ к файлам в этой папке пользователям на уровне веб-сервера.

```
<handlers>
<remove name="BlockViewHandler"/>
<add name="BlockViewHandler"
  path="*" verb="*" preCondition="integratedMode"
  type="System.Web.HttpNotFoundHandler"/>
</handlers>
```

Соотнесение представлений и методов контроллеров описано далее в этой главе.

Файл Default.aspx

Файл Default.aspx создается в качестве заглушки, для того чтобы веб-сервер вызывал инфраструктуру ASP.NET при обращении к сайту без указания пути (например, **http://www.remix.ru/**). Сама по себе страница Defaut.aspx пуста и содержит в обработчике события загрузки страницы код, который переадресует вызов инфраструктуре ASP.NET.

```
public void Page_Load(object sender, System.EventArgs e)
{
 string originalPath = Request.Path;
```

```
 HttpContext.Current.RewritePath(Request.ApplicationPath, false);
 IHttpHandler httpHandler = new MvcHttpHandler();
 httpHandler.ProcessRequest(HttpContext.Current);
 HttpContext.Current.RewritePath(originalPath, false);
}
```

Файл Global.asax

Файл Global.asax используется для создания таблицы маршрутизации, используемой для соотнесения запросов к MVC-приложению и конкретных методов контроллеров и параметров вызова этих методов, поскольку файл Global.aspx предоставляет возможность создания обработчиков глобальных событий уровня всего веб-приложения. При запуске приложения задается таблица маршрутизации, а также на этом этапе могут быть выполнены другие операции, о которых можно будет узнать в следующих главах книги.

```
protected void Application_Start()
{
 RegisterRoutes(RouteTable.Routes);
}
```

Метод `RegisterRoutes` описан подробнее далее в этой главе.

Файл Web.config

Файл Web.config описывает конфигурацию приложения, именно в конфигурации описаны модули и обработчики, которые позволяют работать MVC Framework.

Основным модулем является модуль маршрутизации, который вызывается для всех запросов и инициирует работу инфраструктуры MVC Framework.

```
<add name="UrlRoutingModule"
 type="System.Web.Routing.UrlRoutingModule, System.Web.Routing,
 Version=3.5.0.0,
 Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
```

Рассмотрев физическую структуру файлов MVC-приложения, перейдем к принципам функционирования и внутреннего устройства компонентов MVC-приложения.

Обработка запросов MVC-приложением

Для того чтобы понять принципы работы компонентов MVC-приложения, необходимо четко понимать схему обработки запросов. К счастью, жизненный цикл запроса для MVC-приложения очень прост (рис. 1.6).

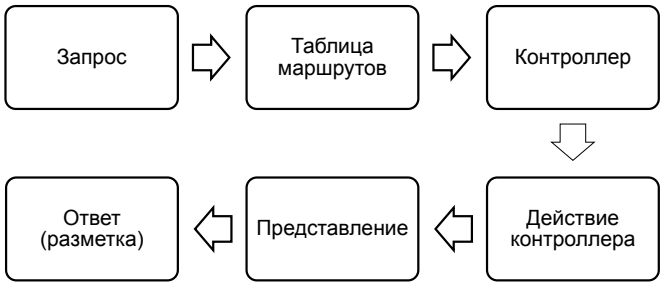


Рис. 1.6. Жизненный цикл запроса для MVC-приложения

Поступающий к веб-серверу HTTP-запрос передается среде выполнения ASP.NET, которая инициализирует инфраструктуру MVC Framewrok и передает запрос для обработки компоненту маршрутизации. На основании *таблицы маршрутизации*, загружаемой при запуске веб-приложения, модуль маршрутизации определяет имена контроллера и метода контроллера, который должен обработать запрос, а также параметры запроса, которые должны быть переданы контроллеру. После этого генерируется контекст запроса, содержащий параметры запроса и среды выполнения приложения (такие как URL запроса, IP-адрес клиента и сервера и т. п.), создается экземпляр класса *контроллера* и ему передается управление путем вызова соответствующего метода класса контроллера — *действия контроллера* в терминах MVC.

Метод контроллера на основании параметров запроса выполняет некоторую логику, выбирает представление, которое должно быть отображено пользователю, и передает управление механизму генерации разметки (*движком представления* в терминах MVC), который уже отображает *представление*.

Для обмена данных между представлением и контроллером используется специальная коллекция `ViewData` — являющаяся основным связующим звеном между контроллером и представлением.

После того как разметка была сгенерирована движком представления, веб-сервер возвращает ее в качестве ответа пользователю по протоколу HTTP. На этом жизненный цикл обработки запроса MVC-приложением заканчивается.

Компоненты MVC-приложения

Рассмотрим подробнее внутреннее устройство таблицы маршрутизации, контроллеров и представлений, для того чтобы продемонстрировать механизм работы MVC-приложения. Подробная информация по каждому из компонентов будет предоставлена в соответствующих главах, посвященных каждому из компонентов, сейчас же нам необходимо посмотреть на состав этих ком-

понентов на очень высоком уровне, чтобы понимать принципы работы MVC-приложений.

Таблица маршрутизации

Таблица маршрутизации определяет набор правил, на основании которых происходит анализ URL-запроса и вычленения из URL информации, определяющей имя контроллера и действия контроллера, а также сопоставление параметров запроса. В проекте-заготовке правила добавляются в методе `RegisterRoutes`, описанном в файле Global.asax (листинг 1.1).

Листинг 1.1. Метод `RegisterRoutes`

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute( "Default",    // Название маршрута
    "{controller}/{action}/{id}",  // URL с параметрами
    new { controller = "Home", action = "Index", id = "" }
                                   // Значения по умолчанию
    );
}
```

Таблица маршрутизации заполняется двумя типами маршрутов — теми, которые должны быть обработаны MVC Framework (метода `MapRoute` коллекции `RouteCollection`), и теми, обработка которых должна быть передана дальше инфраструктуре ASP.NET в обход механизмов MVC Framework (метод `IgnoreRoute` коллекции `RouteCollection`).

В случае с игнорируемыми маршрутами задается определенный адрес либо маска. Так, приведенный в листинге 1.1 маршрут исключает запросы к файлам с расширением axd (используются инфраструктурой ASP.NET для встроенных ресурсов).

Маршруты, обрабатываемые MVC Framework, задаются набором параметров: названием маршрута для идентификации в коллекции, описанием шаблона URL и набором значений по умолчанию. Среди всех параметров обязательными являются `controller` — указывающий имя контроллера, обрабатывающего запросы, удовлетворяющие маске, и `action` — указывающий действие контроллера, обрабатывающего запрос. Все остальные параметры задаются произвольно, и их имена используются для передачи значений при вызове методов контроллера.

Контроллер

Рассмотрим код контроллера `HomeController`, приведенный в листинге 1.2.

Листинг 1.2. Код контроллера `HomeController` из приложения-заготовки

```
[HandleError]
public class HomeController : Controller
{
    public ActionResult Index()
   {
       ViewData["Message"] = "Welcome to ASP.NET MVC!";
       return View();
    }

    public ActionResult About()
    {
       return View();
    }
}
```

На примере этого кода можно рассмотреть несколько основных концепций. Прежде всего, все классы контроллеров наследуют тип `Controller`, предоставляющий инфраструктуру для обработки запросов.

Каждый из контроллеров содержит методы, возвращающие значения типа `ActionResult`. В приведенном примере используется вспомогательный метод `View`, возвращающий тип `ViewResult`, который наследует тип `ActionResult` и передает управление механизму представлений — если параметр "имя представления" не передан методу `View`, то используется имя действия в качестве имени представления.

Задачей контроллера является обработка параметров запроса (в примере параметров действия не принимают, что тоже может быть в реальных приложениях — для отображения страниц, не зависящих от параметров запроса), заполнение коллекции `ViewData` и передача управления движку представлений.

В данном случае, например, действие `Index` поместит в коллекцию `ViewData` элемент `Message`, после чего передаст управление представлению Index.aspx, расположенному в папке /Views/Home.

Кроме того, в листинге 1.2 проиллюстрирована еще одна важная концепция, касающаяся кода контроллеров, — использование атрибутов для управления поведением контроллеров. Так, для того чтобы управлять поведением всего контроллера, можно установить атрибуты на класс контроллера, а для того

чтобы управлять поведением конкретного действия — на соответствующий метод класса контроллера. В приведенном в листинге 1.2 коде используется атрибут для всего класса контроллера `HandleError`, который инструктирует среду MVC Framework на предмет возникновения необработанного исключения в любом из методов контроллера `HomeController`, необходимо будет это исключение поймать и отобразить пользователю специальную страницу с сообщением об ошибке.

Представление

В MVC Framework используется представление на основе ASPX-файлов. Так, например, рассмотрим представление Index.aspx из примера выше. Этому представлению передается коллекция `ViewData` с элементов `Message`, значение которого должно быть отображено на странице. В листинге 1.3 приведен код этого представления.

Листинг 1.3. Представление Index.aspx

```
<%@ Page Language="C#" MasterPageFile="~/Views/Shared/Site.Master"
Inherits="System.Web.Mvc.ViewPage" %>

<asp:Content ID="indexTitle"
 ContentPlaceHolderID="TitleContent" runat="server">
Home Page
</asp:Content>

<asp:Content ID="indexContent"
 ContentPlaceHolderID="MainContent" runat="server">
<h2><%= Html.Encode(ViewData["Message"]) %></h2>
 <p>To learn more about ASP.NET MVC visit
 <a href="http://asp.net/mvc"
   title="ASP.NET MVC Website">http://asp.net/mvc</a>.
 </p>
</asp:Content>
```

Как видно из листинга 1.3, в представлении используется шаблон `Site.Master` и метки `Content` для определения содержимого блоков `ContentPlaceHolder`, определенных в `Site.Master`.

Для отображения данных из коллекции `ViewData` используется серверная вставка вида `<%= %>`, с помощью которой можно отобразить значение на странице.

Подробнее о работе с представлениями и создании сложных представлений можно узнать в *главе 5*.

Подход к разработке MVC-приложений

Исходя из внутреннего устройства MVC-приложений, процесс разработки удобно построить по следующей схеме:

1. Создать модель — создать схему базы данных, по схеме базы данных создать логические структуры данных, с которыми будет работать приложение.
2. Описать физическую структуру приложения — задать маршруты, которые будут определять взаимодействие пользователя с приложением.
3. Создать контроллеры и их действия — на основании структуры приложения.
4. Создать представления — для каждого из действий контроллеров создать представления, учитывая возможность вынесения повторяющихся элементов в частичные представления и шаблоны.
5. Разработать модульные тесты для тестирования логики представления, если планируется модификация логики в процессе поддержки и развития приложения.

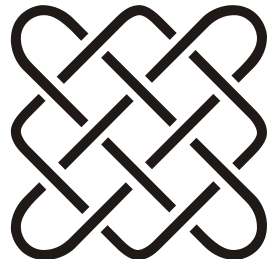
Процесс разработки может быть несколько модифицирован, если разрабатываются веб-приложения с богатой клиентской функциональностью. В случае если создается страница, использующая асинхронные вызовы для обращения к серверу и обновления фрагментов страниц, то может быть удобным изначально создать лишь базовые действия контроллеров, отображающие страницы, после этого в процессе разработки клиентского интерфейса дорабатывать методы, отвечающие на асинхронные запросы. Этот подход будет рассмотрен в *главе 7*, посвященной клиентскому программированию в MVC Framework.

В любом случае, при разработке приложений, вне зависимости от используемой платформы, технологии и парадигмы разработки, необходимо тщательное проектирование и детальное планирование разработки задолго до начала проекта. Хорошее планирование зачастую позволяет существенно сократить сроки разработки проекта, исключив неприятные моменты вроде необходимости рефакторинга части написанного кода для полноценной реализации функциональности.

Заключение

В этой главе мы мельком посмотрели на основные принципы устройства и работы MVC-приложений, а также рассмотрели процесс разработки MVC-приложений. Вооружившись этими знаниями, мы можем перейти к детальному изучению компонентов MVC-приложения в последующих главах этой книги.

ГЛАВА 2



MVC Framework и WebForms

Любая технологическая платформа предлагает разработчику определенные стиль и подходы к разработке приложений. При условии расширяемости платформы и достаточном опыте ее использования разработчику не составит большого труда самостоятельно реализовать любой желаемый подход к разработке веб-приложений, тем не менее в большинстве случаев использование собственных возможностей платформы является наиболее предпочтительным с точки зрения скорости разработки и простоты дальнейшей поддержки решения.

Известный факт, что возможность выбора порождает проблему этого самого выбора — поэтому с появлением на платформе ASP.NET подхода к созданию веб-приложений MVC Framework у разработчиков возникает логичный вопрос — какой подход выбрать и какой подход будет наиболее оправдан при создании очередного веб-приложения. Не хотелось бы расстраивать читателя, жаждущего быстрого ответа на вопрос, какой подход избрать — WebForms или MVC Framework, но все же придется это сделать — ответом на вопрос, стоит ли выбрать MVC Framework для вашего следующего веб-приложения, является вся эта книга.

Существует необозримое множество технологий (и для разных задач лучше подходит та или иная технология) и сделать правильный выбор позволяет только достаточное знание возможностей, преимуществ и недостатков рассматриваемых технологий.

В этой главе проводится краткое сравнение технологий MVC Framework и WebForms, демонстрируется реализация подхода MVC на основе WebForms, что может быть полезно разработчикам, знакомым с WebForms, для того чтобы быстро освоиться с концепцией MVC, а также предлагаются некоторые советы по выбору той или иной технологии.

Сравнение WebForms и MVC Framework

Для того чтобы дать рекомендации по выбору той или иной технологии разработки веб-приложений, рассмотрим сильные и слабые стороны каждой из этих технологий, а также основные принципы, на которых основаны эти технологии.

Технология WebForms

Технология WebForms в ASP.NET была создана для того, чтобы сделать веб-разработку максимально простой для разработчика, знакомого с созданием клиентских приложений на Windows-платформе. По большому счету, создание WebForms мало отличается от создания настольных приложений — элементы управления, использующие механизмы обработки пользовательских действий и хранения состояния, позволяют применять для разработки визуальный подход. Разработчику достаточно разместить элементы управления на странице и определить логику их совместной работы, без необходимости глубокой работы с разметкой HTML и стилями CSS.

Технология WebForms значительно снизила "порог входа" в веб-разработку. Как в свое время появление визуальных средств разработки вроде Visual Basic дало возможность разработчикам разного уровня создавать приложения различной сложности, так появление технологии WebForms привело к быстрому росту количества динамических веб-приложений. Прежде всего, за счет возможности быстрого изучения технологии и быстрого старта разработки полнофункциональных приложений.

Модель расширения WebForms позволила большому количеству компаний разрабатывать собственные элементы управления, которые позволили существенно упростить и ускорить разработку веб-приложений. Кроме того, декларативный подход технологии WebForms позволяет очень просто преобразовывать классические веб-приложения для использования новейших технологий, таких как Ajax и Silverlight.

Преимущества WebForms

- **Высокая скорость разработки.** Благодаря декларативному использованию компонентов и визуальному созданию веб-страниц, достигается высокая скорость разработки функционала веб-приложений. Использование готовых компонентов, интегрируемых между собой, позволяет очень быстро создать работающий прототип приложения, которое в дальнейшем дорабатывается до желаемого результата.
- **Большое количество готовых компонентов.** В состав .NET Framework входит несколько десятков компонентов для реализации наиболее частых

сценариев, а готовые компоненты третьих компаний позволяют сократить время на разработке и отладке собственных решений.

- **Богатая поддержка средствами разработки.** Поддержка WebForms визуальными инструментами разработки, такими как Visual Studio и Expression Web, предоставляет возможность визуального проектирования интерфейса и создания базовых связей между элементами управления. Значительная часть разработки интерфейса приложения может быть выполнена веб-дизайнером, без привлечения разработчика.
- **Автоматическое управление состоянием.** Состояние элементов управления на стороне клиента поддерживается автоматически, и разработчику нет необходимости отслеживать инициализацию всех элементов управления страниц между отправками данных на сервер (postback).
- **Событийная модель элементов управления.** Механизмы обработки клиентских событий позволяют создавать обработчики событий на стороне сервера, разбивая общий код логики взаимодействия веб-страницы с пользователем на небольшие логические блоки.
- **Высокая степень абстракции над HTML-разметкой.** Разработчику нет необходимости работать со страницей на уровне HTML-разметки. Элементы управления создают необходимую разметку во время генерации разметки для конечного пользователя. Кроме того, абстракция над разметкой позволяет генерировать разметку в зависимости от браузера, используемого для доступа к веб-приложению, что особенно важно для приложений, используемых на мобильных устройствах.
- **Простота изучения технологии.** Общий набор концепций, делающий разработку веб-приложений похожей на разработку настольных приложений, и высокая степень абстракции над нижележащими технологиями позволяют быстро приступить к созданию веб-приложений.
- **Возраст технологии.** В современном быстро меняющемся мире десяток лет — это уже очень много. За время существования WebForms было выработано множество методик создания веб-приложений, продуманы различные сценарии использования технологии и накоплен большой опыт в сообществах разработчиков.

Недостатки WebForms

- **Связка разметки и логики страницы.** Разметка страницы, определенная в ASPX-файле, жестко привязана к коду логики, определенному в code-behind-файле, поскольку на основании разметки генерируется частичный класс, являющийся частью класса, определенного в code-behind-файле. В терминах MVC получается, что логика компонентов контроллера и

представления смешиваются. Особенно в случае использования элементов, декларативно работающих с данными, таких как SqlDataSource/ DataGrid, смешивается бизнес-логика, логика интерфейса и работы с данными.

- **Отсутствие полного контроля над конечной разметкой страницы.** Поскольку в технологии WebForms генерация HTML-кода отдана на уровень элементов управления, разработчик не имеет полного контроля над финальной разметкой страницы, загружаемой в браузер пользователя. В большинстве случаев это не является проблемой, пока не возникает необходимости создавать большое количество клиентского кода, работающего с моделью документа HTML-страницы — в этом случае создание обходных путей может занять недопустимо много времени.
- **Сложность тестирования логики приложения.** Поскольку есть связка интерфейса с бизнес-логикой, тестирование логики приложения в отрыве от интерфейса затруднено, что усложняет автоматическое тестирование и требует использования инструментов, позволяющих эмулировать действия пользователя на интерфейсе приложения. Это становится критичным, когда для выполнения тестов через пользовательский интерфейс требуется неоправданно большое время, а для успешного тестирования необходимо частое повторное проведение тестов.
- **Неестественная для веб-среды модель сохранения состояния.** Модель событий и хранения состояния, являющаяся преимуществом WebForms и позволяющая создавать веб-приложения по образу и подобию настольных приложений, является и недостатком этой модели. HTTP-протокол не поддерживает состояния и, при попытке это состояние добавить в жизненный цикл приложения, приходится создавать нагромождения дополнительного слоя логики, отвечающего за работу с состоянием. При использовании большого количества элементов на странице скрытое поле ViewState, используемое для хранения состояния между отправками данных формы на сервер, может достигать больших размеров, существенно влияя на скорость передачи данных между сервером и клиентом. В этом случае разработчику приходится приложить дополнительные усилия по оптимизации хранимых данных о состоянии, использовать другие механизмы для хранения состояния и так или иначе вмешиваться в стандартную модель WebForms.

Технология MVC Framework

В *главе 1* были перечислены некоторые преимущества архитектурного подхода MVC и реализации в MVC Framework. Здесь же мы подробнее остановимся на достоинствах и недостатках MVC Framework в сравнении с WebForms.

Преимущества MVC Framework

- **Четкое разделение логических слоев.** Отделение представления от контроллера предоставляет возможность простой замены движка представления без модификации кода контроллера, независимость от реализации модели позволяет описать только интерфейсы объектов и подменять реализацию при необходимости.
- **Полный контроль над разметкой.** Возможность получить "чистый" HTML-код значительно упрощает разработку и поддержку клиентского JavaScript-кода.
- **Логическое разделение функциональности.** В веб-приложениях MVC есть четкое разделение на действия контроллеров, и каждое действие имеет собственный URI. В случае WebForms страница может инкапсулировать различную логику работы страницы и сохранять тот же самый URI, например Default.aspx, при выполнении операций со страницей. В MVC-приложении разные действия, инициируемые страницей, соответствуют различным действиям контроллера. Так, например, метод `Edit` контроллера возвращает страницу, а метод `Update` используется для обновления данных и отображения сообщения об изменении данных.
- **"Красивые" URL-адреса.** Поскольку MVC Framework предполагает использование гибкой системы маршрутизации, пользователь получает удобные и понятные URL-адреса страниц в виде /Products/List, /Products/Edit/1 и т. п.
- **Прозрачный процесс обработки запроса.** Жизненный цикл страницы в WebForms описывается механизмом событий, этот механизм легко расширяется и позволяет добавлять расширенную функциональность в страницу без модификации кода самой страницы — через глобальные обработчики HttpModule на события той или иной страницы может быть подписан внешний обработчик. Однако в больших и сложных системах наличие множества обработчиков страницы делает процесс обработки запроса нелинейным и сложным для отладки. Кроме того, для успешной работы со страницами в WebForms разработчику нужно очень хорошо представлять последовательность выполнения действий по инициализации инфраструктуры страницы, для того чтобы вносить необходимые модификации в общее течение процесса обработки страницы. В случае MVC Framework процесс обработки запроса легко прослеживается, поскольку обработка запроса разделена на небольшое количество очень простых шагов.
- **Поддержка различных движков генерации разметки.** Компоненты представления, входящие в поставку MVC Framework, могут быть легко заменены на написанные самостоятельно либо разработанные другими разработчиками. В проекте MVCContrib (**http://www.codeplex.com/**

MVCContrib) доступны альтернативные движки представления, такие как Brail, NHaml, NVelocity и XSLT.

- **Полноценная поддержка всех возможностей платформы ASP.NET.** Применяя MVC Framework, разработчик может использовать все возможности платформы ASP.NET точно так же, как и разработчик, использующий WebForms. MVC Framework не ограничивает использование стандартных служб ASP.NET, таких как Session, Cache, Membership, Profile, Roles и т. п.

Недостатки MVC Framework

- **Более высокий "порог входа" в технологию.** Разработчику, только начинающему осваивать создание веб-сайтов, потребуется достаточно подробное изучение веб-технологий, таких как HTML, CSS и JavaScript, для успешного создания многофункциональных веб-приложений. Если технология WebForms во многом "экранирует" разработчика от веб-среды, предоставляя высокую степень абстракции над веб-страницами, то MVC Framework сразу же требует достаточно глубокого понимания веб-технологий.
- **Отсутствие механизма хранения состояния.** MVC Framework не предлагает механизма хранения состояния, и разработчику необходимо реализовывать его самостоятельно, используя скрытые поля, cookie-файлы или хранение данных в URL.
- **Сложности создания библиотек компонентов.** Отсутствие механизма элементов управления, присутствующего в WebFroms, затрудняет создание компонентов для повторного использования. При существующем подходе к созданию представлений инкапсуляция полной логики повторно используемого компонента в отдельную сборку затруднена.
- **Молодость технологии.** Поскольку релиз MVC Framework вышел всего за несколько месяцев до момента написания этих строк, сообщество разработчиков не накопило еще того опыта, который накоплен с WebForms, а компании-разработчики компонентов еще не создали такого большого количества расширений для MVC Framework, как в случае с WebForms.

Выбор подхода к разработке веб-приложения

На платформе ASP.NET можно создать любое веб-приложение, с любым уровнем сложности и возможностями по масштабированию, несмотря на выбранный подход, однако правильный выбор подхода позволит достичь результата меньшими усилиями и упростить дальнейшую поддержку проекта. Приступая к проектированию очередного веб-приложения, вы, возможно, задумаетесь о выборе подхода WebForms или MVC Framework.