ГЛАВА 7

Общее представление о проектах ASP.NET MVC

Пройдя все этапы построения реалистичного приложения MVC под названием SportStore, вы попутно приобрели изрядный объем знаний о разработке на платформе ASP.NET MVC. Однако это был лишь один пример, не охватывающий всех средств и возможностей MVC Framework. Для того чтобы восполнить недостающее, мы предложим более систематическое описание каждого аспекта MVC Framework. В главе 8 будет рассматриваться базовая система маршрутизации. В главе 9 будет продемонстрированы возможности, доступные для построения контроллеров и действий. В главе 10 внимание будет сосредоточено на встроенном механизме представлений ASP.NET MVC. В остальных главах будут описаны другие задачи и сценарии веб-разработки, в том числе вопросы, связанные с безопасностью и развертыванием.

Чтобы не упустить из виду даже мельчайшие детали каждого компонента MVC, сначала окинем взглядом общую картину. В этой главе мы рассмотрим общее устройство приложений MVC: структуру проекта по умолчанию и соглашения по именованию, которым необходимо следовать. Вы получите полное представление о всем процессе обработки запросов и узнаете, как все компоненты платформы работают вместе.

Разработка приложений MVC в Visual Studio

Во время установки ASP.NET MVC выполняются следующие действия.

- В глобальном кэше сборок (GAC) регистрируется сборка MVC Framework под названием `System.Web.Mvc.dll`, а ее копия помещается в каталог `\Program Files\Microsoft ASP.NET\ASP.NET MVC 1.0\Assemblies`.
- В папку `\Common7\IDE`, через которую ASP.NET MVC интегрируется в Visual Studio, устанавливаются разнообразные шаблоны. Вот что включают эти шаблоны.
  1. Шаблоны проектов для построения новых веб-приложений ASP.NET MVC и тестовых проектов (в подпапках `ProjectTemplates\CSharp\Web\1033` и `Test`).
  2. Шаблоны элементов для создания контроллеров, представлений, частичных представлений и мастер-страниц через пункт меню Add Item (Добавить элемент) (в подпапке `ItemTemplates\CSharp\Web\MVC`).
  3. Шаблоны T4, генерирующие код для предварительного заполнения контроллеров и представлений при их создании через пункты меню Add Controller (Добавить контроллер) и Add View (Добавить представление) (в подпапке `ItemTemplates\CSharp\Web\MVC\CodeTemplates`).

- Добавляется набор файлов сценариев для регистрации файлового расширения `.mvc` на сервере IIS, в случае, если планируется его использование (в папке `\Program Files\Microsoft ASP.NET\ASP.NET MVC 1.0\Scripts`). Однако обычно вам эти сценарии не применяются, потому что расширение `.mvc` обычно не должно появляться в URL. Если все-таки это понадобится, расширения URL можно зарегистрировать в диспетчере IIS Manager с графическим интерфейсом, как будет показано в главе 14.

При желании можно отредактировать шаблоны Visual Studio и внесенные изменения отразятся в IDE-среде. Однако для редактирования доступны только шаблоны T4, генерирующие код, и вместо того, чтобы редактировать глобальные шаблоны, централизованно представленные в Visual Studio, имеет больше смысла редактировать специфичные для проекта копии, которые можно поместить в систему управления исходным кодом. Подробнее о механизме шаблонов T4 и его применении в проектах ASP.NET MVC можно узнать на сайте по адресу `http://tinyurl.com/T4mvc`.

Структура стандартного проекта MVC

Когда в Visual Studio создается совершенно новый проект веб-приложения ASP.NET MVC, предоставляется начальный набор папок и файлов, показанный на рис. 7.1. Некоторые из этих элементов играют специальные роли, жестко закодированные в MVC Framework (и подчиняются предопределенным соглашениям об именовании), в то время как другие имеют характер простых рекомендаций относительно структуры проекта. Эти роли и правила описаны в табл. 7.1.

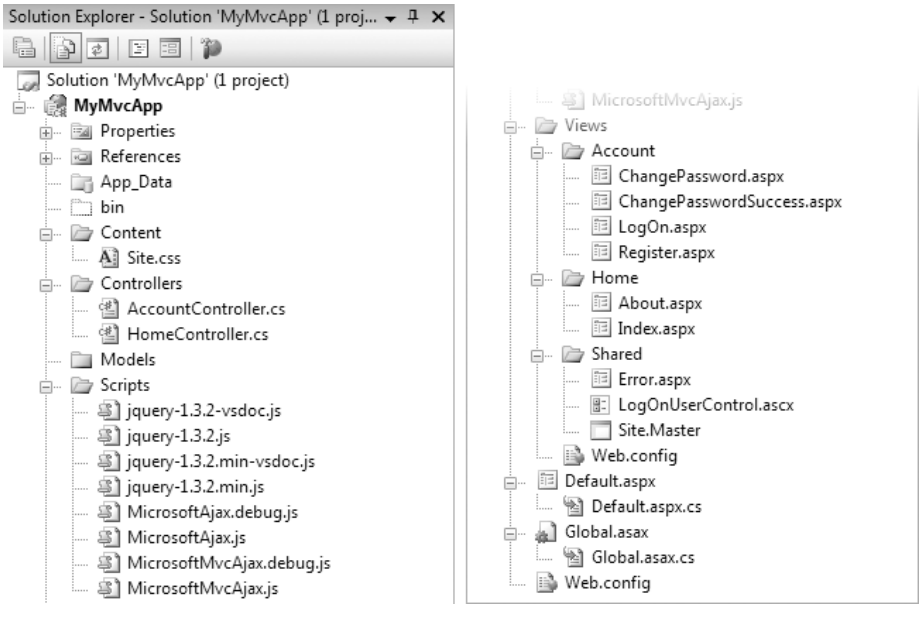

Рис. 7.1. Окно Solution Explorer сразу после создания нового приложения ASP.NET MVC

Таблица 7.1. Файлы и папки в стандартном шаблоне веб-приложения ASP.NET MVC

Папка или файл	Предполагаемое назначение	Специальные полномочия и ответственность
`/App_Data`	Если используется файловая база данных (например, файл `*.mdf` для SQL Server Express Edition или файл `*.mdb` для Microsoft Access), она размещается именно в этой папке. Сюда также можно поместить другие файлы частных данных (например, `*.xml`), поскольку IIS не не обслуживает файлы из этой папки. Тем не менее, можно получать доступ к ним из кода. Файловые базы данных на основе SQL не могут использоваться ни с одной из полноценных версий SQL Server (отличных от Express Edition), поэтому на практике они применяются редко.	Сервер IIS не обслуживает содержимое этой папки для внешнего мира. Если в системе установлена версия SQL Server Express Edition, а строка подключения содержит `AttachDbFileName=|DataDirectory|MyDatabase.mdf`, будет автоматически создана и подключена файловая база данных `/App_Data/MyDatabase.mdf`.
`/bin`	Здесь находится скомпилированная сборка .NET веб-приложения MVC и все прочие сборки, на которые она ссылается (как в традиционном приложении ASP.NET WebForms).	Сервер IIS рассчитывает здесь найти ваши сборки DLL. Во время компиляции Visual Studio копирует в эту папку все сборки DLL, на которые производятся ссылки (за исключением тех, что находятся в глобальном кэше сборок (GAC)). Сервер IIS не обслуживает содержимое этой папки для внешнего мира.
`/Content`	Это место для статических, публично доступных файлов (например, `*.css` и изображения).	Отсутствуют; это просто рекомендация. При желании эту папку можно удалить, но все равно нужно где-то хранить изображения и файлы CSS, и данная папка — вполне подходящее место.
`/Controllers`	Здесь находятся классы контроллеров (т.е. классы, унаследованные от `Controller` или реализующие `IController`).	Отсутствуют; это просто рекомендация. Не имеет значения, поместите вы контроллеры непосредственно в эту папку, в ее подпапку или куда-то в другое место проекта, потому что все они компилируются в одну сборку. Классы контроллеров также могут быть помещены в другие ссылаемые проекты или сборки. Первоначальное содержимое этой папки можно удалить (`HomeController` и `AccountController`) — они просто демонстрируют, с чего можно начать.
`/Models`	Это место для размещения классов, представляющих модель предметной области. Однако во всех приложениях, за исключением наиболее простых, модель предметной области лучше помещать в отдельный проект библиотеки классов C#. В этом случае папку `/Models` можно либо удалить, либо использовать не для полноценных моделей предметной области, а для простых презентационных моделей, которые существуют только для передачи данных от контроллеров к представлениям.	Отсутствуют; эту папку можно удалить.

Папка или файл	Предполагаемое назначение	Специальные полномочия и ответственность
`/Scripts`	Это еще одно место для размещения статических, публично доступных файлов, но предназначенное в основном для файлов кода JavaScript (`*.js`). Файлы `Microsoft*.js` предназначены для поддержки вспомогательных методов ASP.NET MVC `Ajax.*`, а `jquery*.js`, необходимы в случае использования библиотеки jQuery (см. главу 12).	Отсутствуют; эту папку можно удалить. Однако если вы планируется использование вспомогательных методов `Ajax.*`, то потребуется ссылаться на файлы `Microsoft*.js`, расположенные в другом месте.
`/Views`	Здесь находятся представления (обычно файлы `*.aspx`) и частичные представления (обычно файлы `*.ascx`).	По соглашению представления для класса контроллера `XyzController` находятся в папке `/Views/Xyz`. Представление по умолчанию для действия `DoSomething()` класса `XyzController` должно быть помещено в `/Views/Xyz/DoSomething.aspx` (или `/Views/Xyz/DoSomething.ascx`, если он представляет элемент управления, а не целую страницу). Если изначально доступные классы `HomeController` или `AccountController` не используются, соответствующие представления можно удалить.
`/Views/Shared`	Здесь находятся шаблоны представлений, которые не ассоциированы с определенным контроллером — например, мастер-страницы (`*.Master`) и любые совместно используемые представления или частичные представления.	Если файл `/Views/Xyz/DoSomething.aspx` (или `.ascx`) не может быть обнаружен, то следующее место, где будет произведен поиск — это `/Views/Shared/DoSomething.aspx`.
`/Views/Web.config`	Это *не* главный файл `web.config` приложения. Он содержит только директиву, инструктирующую веб-сервер не обслуживать файлы `*.aspx` из папки `/Views` (так как они должны быть визуализированы контроллером, а не вызываться непосредственно как файлы `*.aspx` в WebForms). Этот файл также содержит конфигурационные настройки, необходимые для того, чтобы стандартный компилятор страниц ASP.NET ASPX правильно работал с синтаксисом шаблонов представлений ASP.NET MVC.	Обеспечивает правильность компиляции и выполнения приложения (как описано в предыдущем столбце).
`/Default.aspx`	Этот файл не имеет особого отношения к ASP.NET MVC, но необходим для совместимости с сервером IIS 6, которому требуется “страница по умолчанию” для сайта. Когда `Default.aspx` выполняется, он просто передает управление системе маршрутизации.	Этот файл удалять нельзя, иначе приложение не будет работать под управлением сервера IIS 6 (хотя на сервере IIS 7, запущенном в режиме Integrated Pipeline (Интегрированный конвейер) все будет в порядке).
`/Global.asax`	В этом файле определен глобальный объект приложения ASP.NET. Его класс отделенного кода (`/Global.asax.cs`) — это место для регистрации конфигурации маршрутизации, а также для установки кода, который будет выполняться при инициализации или останове приложения либо при возникновении необработанного исключения. Работает в точности, как файл `Global.asax` из ASP.NET WebForms.	ASP.NET ожидает найти файл с этим именем, но не обслуживает его для внешнего мира.
`/Web.config`	В этом файле определена конфигурация приложения. Далее в этой главе мы еще поговорим об этом важном файле.	ASP.NET (и IIS 7) ожидает найти файл с этим именем, но не обслуживает его для внешнего мира.

На заметку! Как будет показано в главе 14, приложение MVC развертывается копированием большей части этой структуры папок на веб-сервер. Из соображений безопасности сервер IIS не обслуживает файлы, полный путь которых включает `web.config`, `bin`, `App_code`, `App_GlobalResource`, `App_LocalResources`, `App_WebReferences`, `App_Data` и `App_Browsers`, потому что в файле `applicationHost.config` определены узлы `<hiddenSegment>`, скрывающие их. (Сервер IIS 6 также их не обслуживает, поскольку в нем имеется расширение ISAPI под названием `aspnet_filter.dll`, в котором жестко закодирована их фильтрация.) Аналогичнsv образом, сервер IIS сконфигурирован на фильтрацию запросов `*.asax`, `*.ascx`, `*.sitemap`, `*.resx`, `*.mdb`, `*.mdf`, `*.ldf`, `*.csproj` и ряда других.

Перечисленные выше файлы создаются автоматически при создании нового веб-приложения ASP.NET MVC. Кроме того, есть и другие папки и файлы, которые имеют специальное назначение для ядра платформы ASP.NET. Все они описаны в табл. 7.2.

Таблица 7.2. Дополнительные файлы и папки, имеющие специальное назначение

Папка или файл	Назначение
`/App_GlobalResources` `/App_LocalResources`	Содержит файл ресурсов, используемый для локализации страниц WebForms. Подробнее о локализации будет рассказано в главе 15.
`/App_Browsers`	Содержит XML-файлы `.browser`, описывающие способ идентификации специфических веб-браузеров и их возможности (например, поддерживают ли они сценарии JavaScript).
`/App_Themes`	Содержит "темы" WebForms (включая файлы `.skin`), которые влияют на визуализацию элементов управления WebForms.

Последние из перечисленных выше файлов являются частью платформы ASP.NET и не являются обязательными для приложений ASP.NET MVC. За дополнительной информацией об этом обращайтесь к документации по ASP.NET.

Соглашения об именовании

Вы должны были заметить, что в ASP.NET MVC предпочтение отдается *соглашениям*, а не *конфигурации*[1]. Это означает, например, что явно конфигурировать ассоциацию между контроллерами и их представлениями не потребуется; вы просто следуете определенным соглашениям об именовании, и все работает. (Честно говоря, вам придется конфигурировать довольно много настроек в файле `web.config`, но это в основном касается сервера IIS и ядра платформы ASP.NET.) Несмотря на то что о соглашениях об именовании уже упоминалось ранее, давайте вспомним основные положения.

- Классы контроллера *должны* иметь имена, заканчивающиеся на `Controller` (например, `ProductsController`). Это жестко закодировано в `DefaultControllerFactory`: если вы не будете соблюдать это соглашение, класс не будет распознан как контроллер, и запросы к нему направляться не будут. Обратите внимание, что при создании собственной фабрики `IControllerFactory` (см. главу 9) следовать этому соглашению не обязательно.
- Шаблоны представлений (`*.aspx`, `*.ascx`) должны располагаться в папке `/Views/`*`имяКонтроллера`*. Не включайте сюда суффикс `Controller` — представления для `ProductsController` должны попасть в `/Views/Products`, а не в `/Views/ProductsController`.

[1] Эта тактика (как и фраза) — одна из знаменитых маркетинговых деклараций Ruby on Rails.

- Представление по умолчанию для метода действия должно называться по имени самого метода действия. Например, представление по умолчанию для действия `List` контроллера `ProductsController` должно располагаться в `/Views/Products/List.aspx`. В качестве альтернативы можно указать имя представления (например, возвращая `View("SomeView")`), после чего будет производиться поиск `/Views/Product/SomeView.aspx`.
- Когда представление по имени `/Views/Products/Xyz.aspx` обнаружить не удается, предпринимается попытка найти `/Views/Products/Xyz.ascx`. Если и она не удается, ищется `/Views/Shared/Xyz.aspx`, а затем `/Views/Shared/Xyz.ascx`. Это значит, что папку `/Views/Shared` можете использовать для хранения всех представлений, разделяемых между несколькими контроллерами.

Все соглашения, касающиеся папок и имен, могут быть переопределены с использованием специального механизма представлений. В главе 10 будет показано, как это делается.

Начальный скелет приложения

Как показано на рис. 7.1, вновь создаваемые проекты ASP.NET MVC не являются совершенно пустыми. Они уже оснащены встроенными контроллерами `HomeController` и `AccountController` и несколькими ассоциированными с ними шаблонами представлений. В стандартные файлы проекта встроена довольно большая часть поведения.

1. Контроллер `HomeController` может визуализировать начальную страницу (Home) и страницу с описанием приложения (About). Эти страницы генерируются с использованием мастер-страницы и темы с голубыми умиротворяющими тонами из файла CSS.
2. Контроллер `AccountController` позволяет посетителям регистрироваться и входить в приложение. Он использует аутентификацию Forms Authentication с cookie-наборами для отслеживания входа каждого пользователя и средство членства (membership) базовой платформы ASP.NET для ведения списка зарегистрированных пользователей. Когда кто-либо в первый раз пытается зарегистрироваться или войти в систему, средство членства создаст “на лету” файловую базу данных SQL Server Express. Если сервер SQL Server Express не установлен или не запущен, средство членства работать не будет.
3. Контроллер `AccountController` также включает действия и представления, которые позволяют зарегистрированным пользователям изменять свои пароли. Для этого также применяется средство членства ASP.NET.

Начальный скелет приложения представляет собой замечательное введение в устройство приложений ASP.NET MVC. Тем, кто только начинает работать с MVC Framework, он помогает увидеть интересные особенности сразу же после создания нового проекта.

Однако, маловероятно, что вы сохраните поведение по умолчанию без изменений. Исключением является ситуация, когда разрабатываемое приложение действительно использует средство членства ASP.NET (описанное более подробно в главе 15) для ведения учета зарегистрированных пользователей. Большинство новых проектов ASP.NET MVC можно начинать с удаления многих из этих файлов, как это делалось в главах 2 и 4.

Отладка приложений MVC и модульные тесты

Приложение ASP.NET MVC можно отлаживать точно так же, как традиционное приложение ASP.NET WebForms. Отладчик Visual Studio 2008 практически не изменился по сравнению с прежними версиями, поэтому если вы хорошо с ним знакомы, можете пропустить этот раздел.

Запуск отладчика Visual Studio

Для запуска отладчика Visual Studio просто нажмите <F5> (или выберите пункт меню Debug⇨Start Debugging (Отладка⇨Начать отладку)). Когда это делается первый раз, будет предложено включить отладку в файле `Web.config` (рис. 7.2).

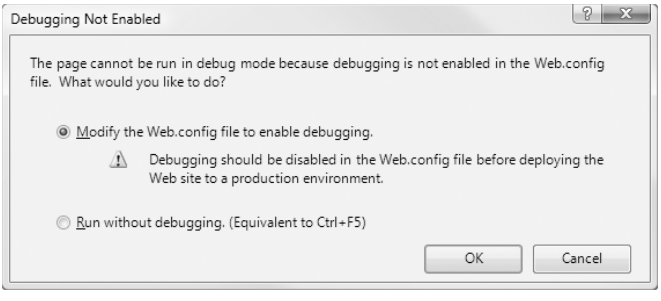

Рис. 7.2. Приглашение Visual Studio включить отладку страниц WebForms

В случае выбора переключателя Modify the Web.config file to enable debugging (Модифицировать файл `Web.config` для включения отладки) Visual Studio обновит узел `<compilation>` файла `Web.config`:

```
<system.web>
  <compilation debug="true">
  ...
  </compilation>
</system.web>
```

Это значит, что шаблоны ASPX и ASCX будут компилироваться с отладочной информацией. Это не повлияет на возможность отладки кода контроллеров и действий, однако в Visual Studio принят именно такой подход. Как показано на рис. 7.3, предусмотрена отдельная настройка, которая влияет на компиляцию файлов `.cs` (например, кода контроллеров и действий) в самом графическом интерфейсе Visual Studio. Режим Debug (Отладка) должен выбираться вручную, так как Visual Studio не делает это автоматически.

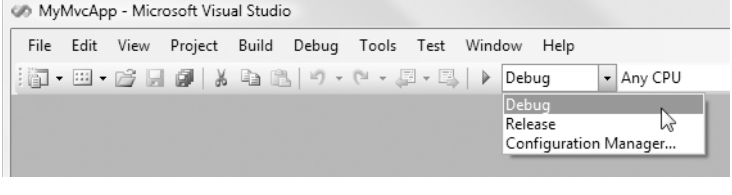

Рис. 7.3. Для использования отладчика проект должен компилироваться в режиме Debug

---

На заметку! Для развертывания на рабочем веб-сервере должен использоваться код, скомпилированный в режиме Release (Выпуск). Вдобавок понадобится установить настройку `<compilation debug="false">` в файле `Web.config` рабочего сайта. Причины этих действий будут подробно описаны в главе 14.

После этого Visual Studio запустит приложение с отладчиком, подключенным к встроенному веб-серверу `WebDev.WebServer.exe`. Нужно будет только установить точку останова, как будет описано ниже (в разделе “Использование отладчика”).

Подключение отладчика к серверу IIS

Если вместо использования встроенного веб-сервера Visual Studio создаваемое приложение запускается на сервере IIS, функционирующем на машине разработки, отладчик можно подключить к IIS. В среде Visual Studio нажмите комбинацию <Ctrl+Alt+P> (или выберите пункт меню Debug⇨Attach to Process (Отладка⇨Присоединиться к процессу)). В открывшемся окне Attach to Process (Присоединиться к процессу), которое показано на рис. 7.4, отыщите работающий процесс по имени `w3wp.exe` (для версии IIS 6 или 7) или `aspnet_wp.exe` (для версии IIS 5 или 5.1). После выбора соответствующего процесса щелкните на кнопке Attach (Присоединиться).

На заметку! Если рабочий процесс найти не удается, возможно, это потому, что вы имеете дело с IIS 7 или работаете через подключение к удаленному рабочему столу (Remote Desktop). В этом случае понадобится отметить флажок Show processes in all sessions (Показать процессы во всех сеансах). Также следует проверить, действительно ли запущен рабочий процесс, открыв приложение в веб-браузере и щелкнув на кнопке обновления в Visual Studio. В среде Windows Vista с включенным средством контроля учетных записей (UAC) система Visual Studio должна быть запущена в режиме *повышения привилегий* (если это не так, то после щелчка на кнопке Attach будет выдано соответствующее предупреждения).

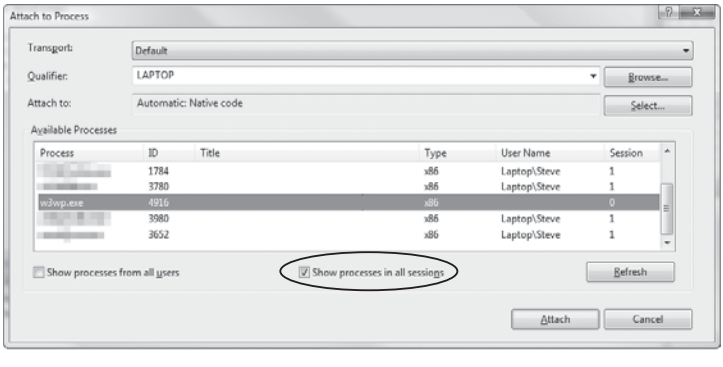

Рис. 7.4. Присоединение отладчика Visual Studio к рабочему процессу IIS 6/7

Подключение отладчика к среде выполнения тестов

При выполнении большого объема модульного тестирования вы обнаружите, что запускаете код в среде выполнения тестов, подобной графической среде NUnit, почти так же часто, как и на веб-сервере. Если тест неожиданно дает сбой (либо вопреки ожиданиям проходит успешно), то вместо сервера IIS отладчик можно подключить к среде выполнения тестов. Удостоверьтесь, что код скомпилирован в режиме Debug, после чего в диалоговом окне Attach to Process (открывающемся по нажатию <Ctrl+Alt+P>) найдите процесс среды выполнения тестов в списке доступных процессов (рис. 7.5).

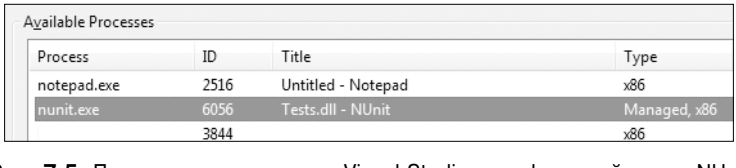

Рис. 7.5. Подключение отладчика Visual Studio к графической среде NUnit

Обратите внимание, что в столбце Type (Тип) показано, какие процессы выполняют управляемый код (т.е. код .NET). С помощью этого столбца можно быстро идентифицировать процессы, выполняющие ваш код.

Удаленная отладка

Если сервер IIS установлен и запущен на других ПК или серверах домена Windows, для которых настроены соответствующие полномочия отладки, можете ввести имя или IP-адрес компьютера в поле Qualifier (Квалификатор) и приступить к удаленной отладке. При отсутствии домена Windows измените значение в раскрывающемся списке Transport (Транспорт) на Remote (Удаленный) и выполняйте отладку по сети (предварительно сконфигурировав на целевой машине монитор удаленной отладки (Remote Debugging Monitor) для ее разрешения).

Использование отладчика

Как только отладчик Visual Studio присоединен к процессу, можно останавливать выполнение приложения и смотреть, что оно делает. Для этого поместите на нужную строку исходного кода точку останова (breakpoint), щелкнув на ней правой кнопкой мыши и выбрав в контекстном меню Breakpoint⇨Insert breakpoint (Точка останова⇨Вставить точку останова), нажав <F9> или щелкнув в серой области слева от строки кода. Возле строки появится кружок красного цвета. Когда присоединенный процесс достигнет этой строки кода, отладчик остановит выполнение, как показано на рис. 7.6.

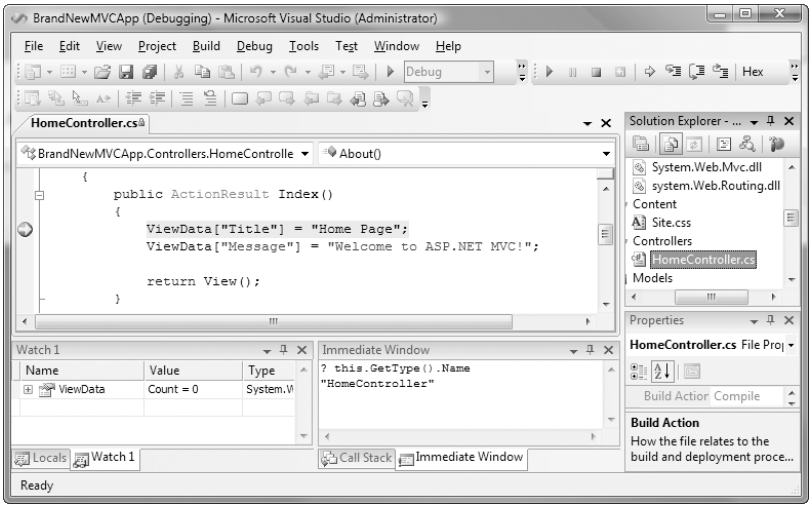

Рис. 7.6. Отладчик достиг точки останова

Отладчик Visual Studio — очень мощный инструмент. Он позволяет читать и модифицировать значения переменных (наводя на них курсор мыши или используя окно Watch (Слежение)), манипулировать потоком выполнения программы (перетаскивая стрелку желтого цвета) или выполнять произвольный код (вводя его в окне Immediate (Немедленное выполнение)). Кроме того, можно просматривать стек вызовов, дизассемблированный машинный код, список потоков выполнения и прочую информацию, отмечая соответствующие пункты в меню Debug⇨Windows (Отладка⇨Окна). За дополнительной справочной информацией по отладчику обращайтесь к специально посвященным этому ресурсам Visual Studio.

Вхождение в исходный код .NET Framework

Существует одно малоизвестное средство отладчика, которое в 2008 г. неожиданно стало весьма удобным. Если приложение обращается к коду посторонней сборки, обычно нет возможности входить в исходный код этой сборки во время отладки (поскольку исходный код просто отсутствует). Однако если поставщик этой сборки опубликует код на *сервере символов* (symbol server), среду Visual Studio можно настроить так, чтобы она извлекала этот код на лету и отображала во время отладки.

С января 2008 г. Microsoft открыла общедоступный сервер символов, содержащий исходный код большинства библиотек .NET Framework. Теперь можно входить в исходный код `System.Web.dll` и других сборок ядра. Это исключительно полезно, когда вы сталкиваетесь с загадочной проблемой, решить которую не помогает даже Google. Сервер символов предоставляет больше информации, чем может обеспечить дизассемблирование с помощью Reflector — вы получаете доступ к оригинальному исходному коду со всеми комментариями (рис. 7.7).

Для настройки такого доступа понадобится среда Visual Studio 2008 SP1. Соответствующие инструкции можно найти по адресу `referencesource.microsoft.com/serversetup.aspx`.

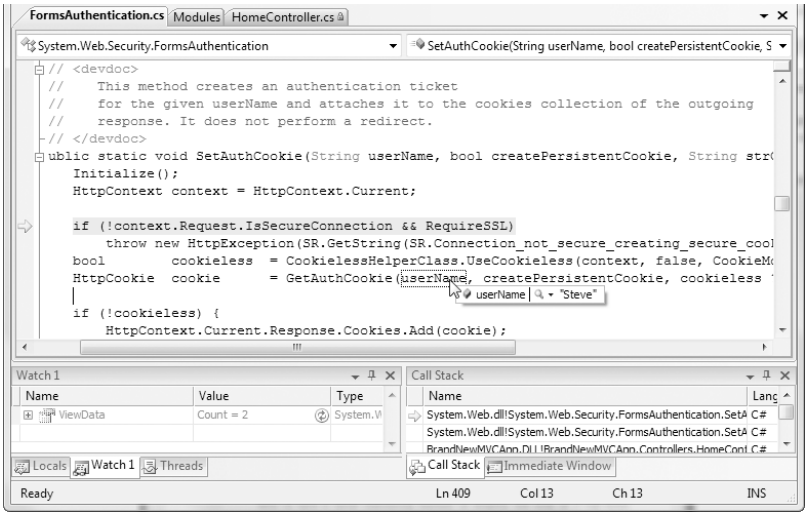

Рис. 7.7. Вхождение в исходный код средства Forms Authentication в Microsoft ASP.NET

На заметку! В Microsoft решили предоставить возможность даже свободно загружать исходный код ASP.NET MVC, что позволяет его компилировать (и модифицировать) самостоятельно. Однако это не касается исходного кода библиотек .NET Framework — доступ к нему можно получить только через сервер символов Microsoft во время отладки. Загрузить исходный код библиотек .NET Framework целиком и скомпилировать самостоятельно нельзя.

Вхождение в исходный код ASP.NET MVC

Поскольку исходный код ASP.NET MVC доступен для свободной загрузки целиком, в решение можно включить исходный код `System.Web.Mvc` (как если бы он был создан вами). После этого во время отладки с помощью пункта меню Go to Declaration (Перейти

к объявлению) в Visual Studio вы сможете переходить по любой ссылке из написанного исходного кода к исходному коду платформы. Это может сэкономить массу времени при определении причин некорректной работы приложения.

Настроить такой доступ нетрудно, если помнить о нескольких возможных проблемах и способах их решения. После выхода в печать этой книги инструкции на этот счет могут претерпеть изменения, поэтому ищите актуальную информацию в блоге автора по адресу `http://tinyurl.com/debugMvc`.

Конвейер обработки запросов

Выше был представлен обзор проектов ASP.NET MVC с точки зрения среды разработки Visual Studio. Теперь давайте рассмотрим, что в действительности происходит во время обработки процессами MVC Framework каждого входящего запроса.

Конвейер обработки запросов ASP.NET MVC сравним с жизненным циклом страницы в ASP.NET WebForms в том смысле, что он отражает внутреннюю организацию системы. Хорошее его понимание совершенно необходимо в ситуациях, когда применяются какие-то нестандартные подходы. В отличие от жизненного цикла традиционной страницы ASP.NET, конвейер MVC чрезвычайно гибок: любую его часть можно модифицировать по собственному усмотрению; можно даже вообще реорганизовать или заменить компоненты. Обычно расширять или изменять этот конвейер *не* требуется, но делать это можно — именно на этом основана всесторонняя расширяемость ASP.NET MVC. Например, при разработке приложения SportStore был реализован специальный фабричный интерфейс `IControllerFactory` для создания экземпляров контроллеров через контейнер IoC.

На рис. 7.8 показано представление конвейера обработки запросов. Центральная вертикальная линия обозначает стандартный конвейер ASP.NET MVC (для запросов, визуализирующих представление), а боковые ответвления символизируют основные точки расширяемости.

На заметку! Чтобы не слишком усложнять картину, на диаграмме не показаны все события и точки расширения. Самое важное, что не показано — фильтры, которые можно внедрять перед и после выполнения методов действий, а также перед и после выполнения результатов действий (включая `ViewResults`). Например, в главе применялся фильтр `[Authorize]` для обеспечения безопасности и контроллера. Дополнительные сведения о том, куда должны подставляться фильтры, вы узнаете далее в этой главе.

В оставшейся части главы приводится более подробное описание конвейера обработки запросов. Затем в главах 8, 9 и 10 каждый из основных компонентов рассматривается по очереди, давая исчерпывающее представление о средствах и возможностях ASP.NET MVC.

Стадия 1: сервер IIS

Информационные службы Интернета (Internet Information Services — IIS), реализующие производственный веб-сервер Microsoft, играют центральную роль в конвейере обработки запросов. При поступлении каждого HTTP-запроса и перед активизацией ASP.NET драйвер устройства Windows режима ядра по имени `HTTP.SYS` анализирует запрошенную комбинацию URL/номер порта/IP-адрес, сопоставляет ее со списком приложений и передает зарегистрированному *приложению* (которым является либо веб-сайт IIS, либо виртуальный каталог внутри веб-сайта IIS).

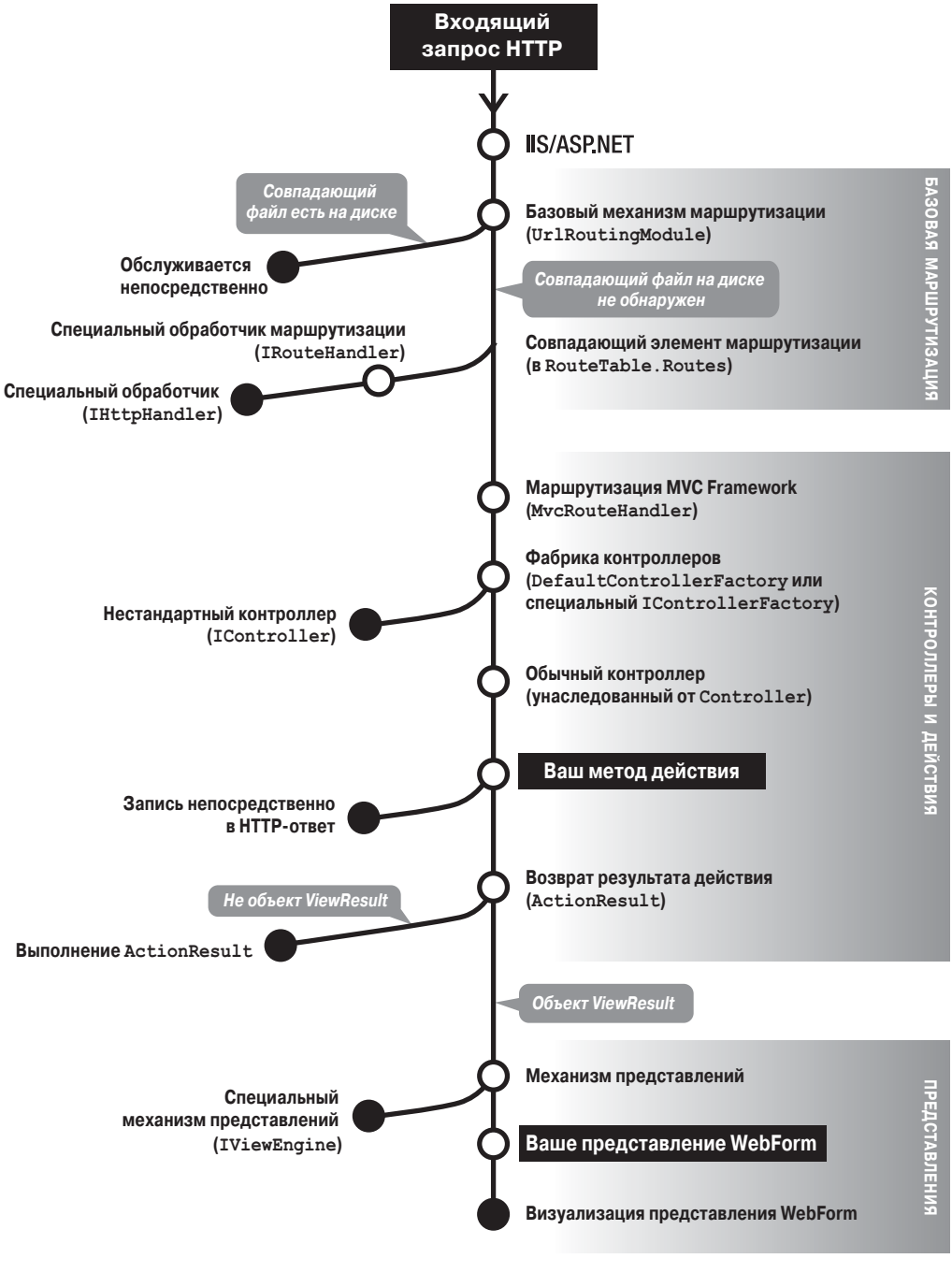

Рис. 7.8. Конвейер обработки запросов ASP.NET MVC

Поскольку приложения ASP.NET MVC построены на основе платформы ASP.NET, она должна быть включена для этого пула приложений IIS (каждое приложение IIS назначается в определенный пул). Включать ASP.NET можно в одном из двух *управляемых режимах конвейера*.

- В *режиме ISAPI*, который также называется *классическим режимом*, ASP.NET вызывается через расширение (`aspnet_isapi.dll`)[2], ассоциированное с определенными расширениями имен файлов в URL (например, `.aspx`, `.ashx`, `.mvc`). В IIS 6 можно настроить *отображение по шаблону*, так что `aspnet_isapi.dll` будет отображать все запросы, независимо от расширения имени файла в URL. Более подробные сведения о развертывании приложений MVC Framework на сервере IIS 6, включая настройку отображения по шаблону, можно найти в главе 14.
- В *интегрированном режиме* (поддерживаемом только в IIS 7+), .NET является естественной частью конвейера обработки запросов IIS, так что никаких расширений ISAPI, ассоциированных с определенным расширением имени файла в URL, не понадобится. Это облегчает использование конфигураций маршрутизации с чистыми URL (т.е. без расширений имен файлов).

В любом случае, как только ASP.NET получает входящий запрос, каждый зарегистрированный модуль HTTP уведомляется о начале нового запроса. (Модуль HTTP — это класс .NET, реализующий интерфейс `IHttpModule`, который можно "подключить" к конвейеру обработки запросов ASP.NET.)

Одним из важных модулей HTTP, регистрируемых по умолчанию в любом приложении ASP.NET MVC, является `UrlRoutingModule`. Этот модуль представляет собой начало базовой системы маршрутизации, которая рассматривается ниже. Наличие узла `<httpModules>` в файле `web.config` говорит о том, что этот модуль зарегистрирован для IIS 6:

```
<system.web>
 <httpModules>
 <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule, ..." />
 </httpModules>
</system.web>
```

Если вы имеете дело с IIS 7, откройте его графический интерфейс Modules (Модули). Для этого дважды щелкните на элементе Администрирование панели управления, затем в открывшемся окне дважды щелкните на элементе Диспетчер служб IIS, в дереве слева выберите свой веб-сайт и затем дважды щелкните на значке Модули. В открывшемся графическом интерфейсе Модули можно отредактировать узел `<system.webServer>/<modules>` из `web.config` (рис. 7.9).

Стадия 2: базовая маршрутизация

Когда модуль `UrlRoutingModule` вовлекается в обработку запроса, запускается система маршрутизации `System.Web.Routing`. Задача маршрутизации состоит в распознавании и разборе произвольных шаблонов входящих URL, с установкой структуры данных *контекста запроса*, которой будут пользоваться последующие компоненты (например, в ASP.NET MVC эта структура служит для передачи управления соответствующему классу контроллера MVC и для передачи параметров методам действий).

[2] API-интерфейс служб Интернета (Internet Services API — ISAPI) — это старая система подключаемых модулей IIS. Расширения ISAPI можно создавать только в неуправляемом коде (например, C/C++).

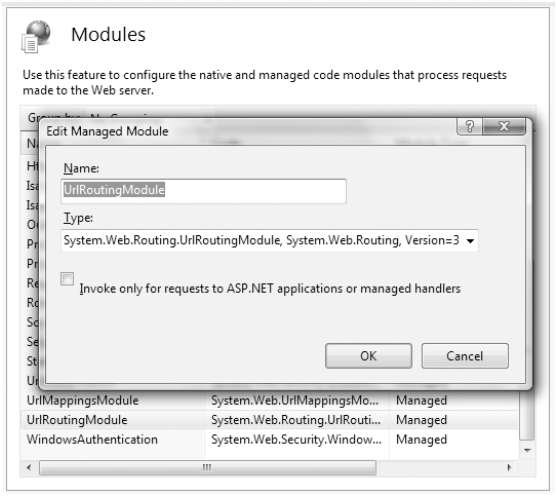

Рис. 7.9. Редактирование модуля `UrlRoutingModule` в графическом интерфейсе Modules

На рис. 7.8 видно, что во время базовой маршрутизации сначала производится проверка, соответствует ли входящий URL какому-то файлу на диске. Если да, то маршрутизация завершается, и дальнейшая обработка этого запроса поручается серверу IIS. Для статических файлов (например, `.gif`, `.jpeg`, `.png`, `.css` или `.js`) это означает, что сервер IIS обработает их естественным образом (потому что они существуют на диске), что очень эффективно. Аналогично, это означает, что традиционные страницы `.aspx` ASP.NET WebForms также будут обработаны обычным образом (поскольку они также присутствуют на диске).

Однако если входящий URL не соответствует никакому файлу на диске (например, запросы к контроллерам MVC, представляющим собой типы .NET, а не файлы на диске), то для выяснения способа обработки входящего URL на стадии базовой маршрутизации анализируется активная конфигурация.

Конфигурация маршрутизации

Конфигурация маршрутизации содержится в статической коллекции по имени `System.Web.Routing.RouteTable.Routes`. Каждый ее элемент представляет собой отдельный шаблон URL с необязательными заполнителями для *параметров* (например, `/blog/{year}/{entry}`) и *ограничениями*, которые лимитируют диапазон допустимых значений каждого параметра. Каждый элемент также специфицирует *обработчик маршрута* — объект, реализующий `IRouteHandler`, — который получает и обрабатывает запрос. Обычно коллекция `RouteTable.Routes` заполняется добавлением кода к методу по имени `RegisterRoutes()` в файле `Global.asax.cs`.

Чтобы сопоставить входящие запросы с определенными элементами `RouteTable.Routes`, система базовой маршрутизации просто начинает с начала коллекции `RouteTable.Routes` и просматривает список вниз, выбирая первый элемент, который соответствует входящему запросу. Найдя такой элемент, маршрутизация передает управление объекту-обработчику, указанному для этого элемента, передавая ему структуру данных "контекста запроса", которая описывает выбранный элемент `RouteTable.Routes`, и все значения параметров, переданные в URL. Здесь вступает в действие MVC Framework, что мы и рассмотрим далее.

Более детальные сведения о системе маршрутизации предлагаются в главе 8.

Стадия 3: контроллеры и действия

Итак, система маршрутизации выбрала определенный элемент `RouteTable.Routes` и разобрала все параметры маршрутизации из URL. Она упаковала эту информацию в структуру, именуемую *контекстом запроса*. Когда же на сцену выходят контроллеры и действия?

Поиск и вызов контроллеров

Для приложений ASP.NET MVC почти все элементы `RouteTable.Routes` специфицируют один определенный обработчик маршрута — `MvcRouteHandler`. Это встроенный в ASP.NET MVC стандартный обработчик маршрутов, служащий мостом между основной маршрутизацией и собственно MVC Framework. Обработчику `MvcRouteHandler` известно, как получить данные контекста запроса и вызвать соответствующий класс контроллера.

Как показано на рис. 7.8, это делается с помощью объекта *фабрики контроллеров*. По умолчанию используется встроенная фабрика по имени `DefaultControllerFactory`, которая при определении корректного класса контроллера для каждого конкретного запроса следует определенным соглашениям об именовании и о пространствах имен. Однако если вы замените встроенный `DefaultControllerFactory` каким-то другим объектом, реализующим интерфейс `IControllerFactory`, или подклассом `DefaultControllerFactory`, то сможете изменить эту логику. Такой подход уже применялся в главе 4 при включении контейнера IoC в конвейер обработки запросов.

Что должны делать контроллеры

Минимальное требование к классу контроллера состоит в том, что он должен реализовать интерфейс `IController`:

```
public interface IController
{
    void Execute(RequestContext requestContext);
}
```

Как видите, интерфейс довольно прост. На самом деле он не специфицирует ничего помимо того, что контроллер должен что-то сделать (т.е. реализовать `Execute()`). Обратите внимание, что параметр `requestContext` предоставляет все данные контекста запроса, сконструированного системой маршрутизации, в том числе параметры, которые извлечены из URL. Кроме того, `requestContext` предоставляет доступ к объектам `Request` и `Response`.

Что обычно делают контроллеры

Чаще всего интерфейс `IController` не реализуется непосредственно. Вместо этого создаваемые классы контроллеров наследуются от `System.Web.Mvc.Controller`. Это встроенный стандартный класс контроллера MVC Framework, который добавляет дополнительную инфраструктуру для обработки запросов. Что более важно, он вводит в систему методы действий. Это значит, что каждый из общедоступных методов класса контроллера может быть вызван через некоторый URL (такие общедоступные методы называются *методами действий*), и это значит, что реализовывать `Execute()` вручную не понадобится.

Хотя методы действий *могут* посылать вывод непосредственно в HTTP-ответ, поступать так не рекомендуется. Из соображений тестируемости и повторного использования кода (о котором речь пойдет ниже) для методов действий лучше возвращать *результат*

действия (объект, унаследованный от `ActionResult`), который описывает желаемый вывод. Например, если необходимо визуализировать представление, вернуть следует `ViewResult`. Каркас MVC затем позаботится о выполнении результата в соответствующий момент в конвейере обработки запросов.

Существует также очень гибкая система *фильтров*. Это атрибуты .NET (например, `[Authorize]`), которыми можно "пометить" класс контроллера или метод действия, внедряя подобным образом дополнительную логику, которая будет выполняться перед или после методов действий либо перед или после выполнения результатов действий. Существует даже встроенная поддержка специальных типов фильтров (фильтров исключений и фильтров авторизации), активизируемых в определенные моменты времени. Фильтры могут появляться в настолько разных местах, что все их даже не удалось показать на рис. 7.8.

Контроллеры и действия (а также связанные с ними средства) образуют несущую конструкцию MVC Framework. Дополнительные сведения о них вы узнаете в главе 9.

Стадия 4: результаты действий и представления

К настоящему времени уже много чего произошло. Давайте подытожим.

- Система маршрутизации сопоставила входящий URL со своей конфигурацией и подготовила структуру данных — контекст запроса. Соответствующий элемент `RouteTable.Route` выбрал `MvcRouteHandler` для обработки запроса.
- Обработчик `MvcRouteHandler` использовал данные контекста запроса с фабрикой контроллеров для выбора и вызова класса контроллера.
- Класс контроллера вызвал один из своих методов действий.
- Метод действия вернул объект `ActionResult`.

В этой точке MVC Framework запросит выполнение объекта `ActionResult` и на этом все. `ActionResult` выполнит то, что должен делать тип `ActionResult` (т.е. вернет строку или данные JSON в браузер, произведет перенаправление HTTP, потребует аутентификации и т.п.). В главе 9 вы узнаете о встроенных типах `ActionResult`, а также о том, как создавать собственные специальные типы.

Визуализация представления

Особое внимание стоит уделить одному подклассу `ActionResult` — `ViewResult`. Он занимается поиском и визуализацией определенного шаблона представления, попутно передавая структуру данных `ViewData`, которая сконструирована методом действия. Это делается вызовом механизма представлений (класса .NET, реализующего `IViewEngine`), назначенного контроллером.

Механизм представлений по умолчанию называется `WebFormViewEngine`. Его шаблоны представлений — это страницы ASPX WebForms (т.е. серверные страницы, используемые ASP.NET WebForms). Страницы WebForms имеют собственный конвейер обработки, который начинает с компиляции "на лету" ASPX/ASCX и проходит через ряд событий, известных под общим названием *жизненный цикл страницы*. В отличие от традиционных WebForms, эти страницы должны быть предельно простыми, потому что при четком разделении ответственности MVC шаблоны представлений не должны делать ничего кроме генерации HTML-разметки. Это означает, что детально разбираться в жизненном цикле страниц WebForms не нужно. Четкое разделение ответственности способствует простоте и сопровождаемости кода.

На заметку! Чтобы разработчики MVC не включали в представления ASPX обработчики событий в стиле WebForms, эти представления обычно вообще не имеют соответствующих классов отделенного кода. Тем не менее, создавать такие классы в файле отделенного кода *можно*, создав в Visual Studio обычную форму Web Form в соответствующем месте представления и затем унаследовав этот класс отделенного кода не от `System.Web.UI.Page`, а от `ViewPage` (или от `ViewPage<T>` для некоторого типа модели `T`). Однако заниматься трюкачеством подобного рода не рекомендуется.

Разумеется, можно реализовать собственный тип `IViewEngine`, полностью заменив механизм представления WebForms. В главе 10 будет дана исчерпывающая информация о представлениях, в частности, о механизме представлений WebForms, а также о некоторых альтернативных и специальных механизмах представлений.

Резюме

В этой главе обзор приложений ASP.NET MVC был представлен с двух точек зрения.

- С точки зрения *структуры проекта* было показано, как работают шаблоны проектов MVC Visual Studio, и каким образом файлы кода организованы по умолчанию. Вы узнали о том, какие файлы, папки и соглашения об именовании являются рекомендованными, а какие — обязательными для данного каркаса. Вы также ознакомились с особенностями отладки приложений ASP.NET MVC в Visual Studio.
- С точки зрения *исполняющей системы* было показано, как ASP.NET MVC обрабатывает входящие HTTP-запросы. Был описан весь конвейер, начиная с определения соответствия маршрута, активизации контроллеров и действий, и заканчивая шаблонами представлений, которые посылают готовую HTML-разметку обратно в браузер. (Это лишь поведение по умолчанию; не существует пределов гибкости в реорганизации конвейера, добавлении, изменении или удалении компонентов. Платформа MVC Framework предлагает разработчикам полный контроль над каждым своим действием.)

Благодаря следующим трем главам ваши пока поверхностные знания превратятся в глубокое и всестороннее понимание каждой составной части ASP.NET MVC. Глава 8 посвящена маршрутизации. В главе 9 описаны контроллеры и действия, а в главе 10 — представления. Вы узнаете обо всех доступных вариантах и о том, как наилучшим способом использовать каждое доступное средство.