ГЛАВА 14

Обзор проектов MVC

Прежде чем погрузиться в детали средств MVC Framework, необходимо рассмотреть ряд дополнительных вопросов. В этой главе представлен обзор структуры и природы приложения ASP.NET MVC, включая стандартную структуру проекта и соглашения об именовании, часть которых необязательны, а часть жестко закодированы в рамках способа функционирования инфраструктуры MVC Framework.

Работа с проектами MVC в Visual Studio

При создании нового проекта ASP.NET среда Visual Studio предоставляет несколько вариантов первоначального содержимого, которое требуется для проекта. Идея состоит в том, чтобы упростить процесс изучения разработчикам-новичкам и применить экономящие время рекомендуемые приемы для распространенных средств и задач. Такая поддержка воплощена через шаблоны, используемые для создания контроллеров и представлений, которые создаются с помощью шаблонного кода для вывода списков объектов данных, редактирования свойств модели и т.д.

В версиях Visual Studio 2013 и MVC 5 разработчики из Microsoft обновили шаблоны и то, что известно как *формирование шаблонов*, чтобы размыть границы между разными видами проектов ASP.NET и предоставить более широкий диапазон шаблонов проектов и конфигураций кода.

Первая часть книги должна была рассеять у вас любые сомнения относительно того, что я не являюсь поклонником подобного подхода к реализации проектов или написанию кода. Намерения, конечно же, благие, однако исполнение, как правило, не вызывает восторга. Одной из характеристик ASP.NET и MVC Framework, которая мне нравится больше всего, является как раз та огромная гибкость, которая позволяет подстроить платформу для удовлетворения моего стиля разработки. Проекты, классы и представления, создаваемые шаблонами Visual Studio, заставляют меня чувствовать, что я ограничен работой в стиле кого-то другого. Вдобавок я нахожу содержимое и конфигурацию чересчур общими и слишком безликими, чтобы приносить пользу. В главе 10 упоминалось, что одна из опасностей применения чувствительного дизайна при ориентации на мобильные устройства, заключается в усреднении, которое приводит к ухудшению пользовательского интерфейса для всех устройств, и нечто подобное происходит с шаблонами Visual Studio. Разработчики в Microsoft не могут знать, какой тип приложения вы должны создать, поэтому они охватывают все основы, но настолько обезличенным и обобщенным способом, что я в любом случае просто отбрасываю стандартное содержимое.

Я предпочитаю начинать с пустого проекта и по мере необходимости добавлять в него папки, файлы и пакеты. Поступая так, вы не только лучше изучите работу MVC Framework, но также будете иметь полный контроль над тем, что содержит ваше приложение.

Однако мои предпочтения не должны влиять на ваш собственный опыт разработки. Вы можете считать шаблоны и формирование шаблонов более удобными, нежели я, особенно если вы делаете только первые шаги в разработке ASP.NET и еще не выработали свой персональный стиль. Вы также можете рассматривать шаблоны проектов как полезный ресурс и источник идей, хотя вы должны проявлять осторожность при добавлении любой функциональности к приложению, пока полностью не разберетесь, как оно работает.

Создание проекта

Когда вы впервые создаете новый проект MVC Framework, вы имеете на выбор две базовых отправных точки: шаблон Empty (Пустой) и шаблон MVC. Эти имена несколько обманчивы, т.к. добавить базовые папки и сборки, требуемые для MVC Framework, можно в любой проект, отметив флажок MVC в разделе Add folders and core references for (Добавить папки и основные ссылки для) диалогового окна New ASP.NET Project (Новый проект ASP.NET), как показано на рис. 14.1. В случае выбора шаблона MVC этот флажок отмечается автоматически.

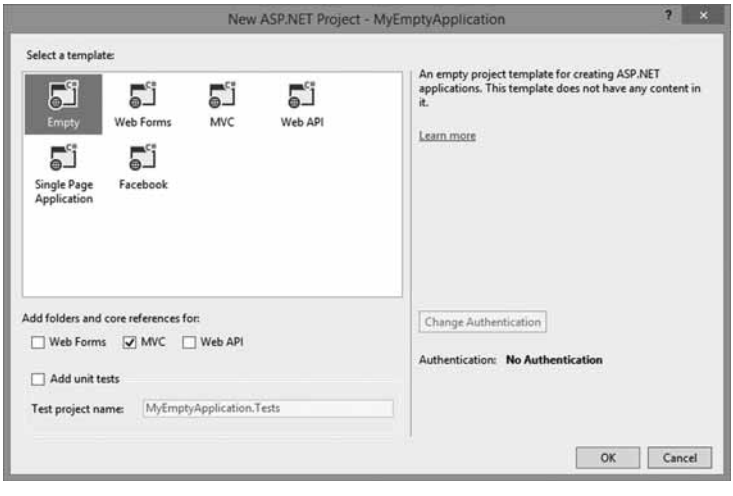

Рис. 14.1. Выбор для нового проекта типа проекта, папок и сборок

Реальное отличие связано с дополнительным содержимым, которое шаблон MVC добавляет в новые проекты. Оно предоставляет готовую отправную точку, включающую стандартные контроллеры и представления, конфигурацию безопасности, ряд популярных пакетов JavaScript и CSS (таких как jQuery и Bootstrap), а также компоновку, которая применяет библиотеку Bootstrap для построения темы, оформляющей пользовательский интерфейс приложения. Вариант Empty проекта содержит только базовые ссылки, требуемые для MVC Framework, и базовую структуру папок. Шаблон MVC добавляет довольно много содержимого, и разницу можно видеть на рис. 14.2, где показано содержимое двух созданных проектов. Проект в левой части создан с применением шаблона Empty при отмеченном флажке MVC. Проект в правой части создан с использованием шаблона MVC, и чтобы продемонстрировать все файлы в нем, пришлось открывать разные папки в окне Solution Explorer (Проводник решения), т.к. список файлов оказался бы слишком длинным.

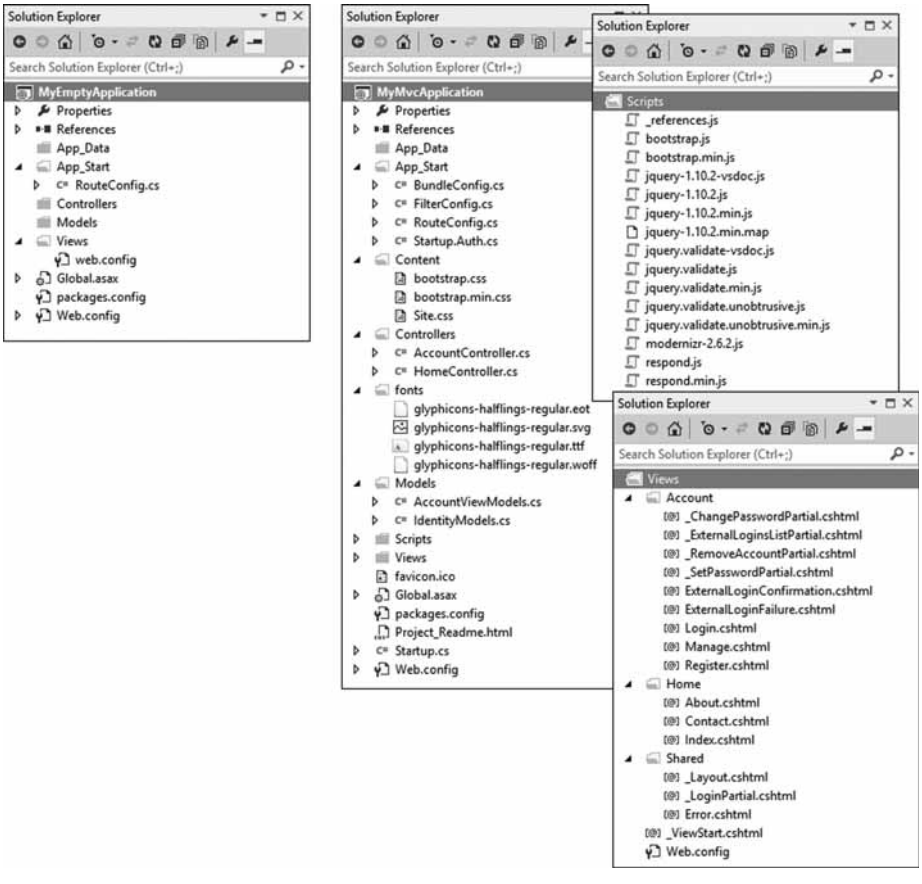

Рис. 14.2. Стандартное содержимое, добавляемое в проект шаблонами Empty и MVC

Дополнительные файлы, добавляемые в проект шаблоном MVC, выглядят хуже, чем есть на самом деле. Некоторые из них относятся к аутентификации, а другие представляют собой файлы JavaScript и CSS, для которых предусмотрены обычные и минимальные версии. (Более подробные сведения будут даны в главе 26.)

Совет. Среда Visual Studio собирает проект, созданный с помощью шаблона MVC, используя пакеты NuGet, а это значит, что вы можете просмотреть, какие пакеты применяются, выбрав пункт Manage NuGet Packages for Solution (Управление пакетами NuGet для решения) в меню Tools⇨Library Package Manager (Сервис⇨Диспетчер библиотечных пакетов). Это также означает возможность добавления тех же самых пакетов в любой проект, в том числе и созданный с использованием шаблона Empty (именно так делалось в примерах, рассмотренных в первой части книги).

Какой бы шаблон вы не выбрали, вы заметите, что результирующие проекты имеют очень похожие структуры папок. Некоторые из элементов в проекте MVC играют особые роли, жестко закодированные в ASP.NET или MVC Framework. Другие имеют отношение к соглашениям об именовании. Все эти основные файлы и папки описаны в табл. 14.1, причем некоторые из них по умолчанию в проектах не присутствуют, но будут введены в последующих главах.

Таблица 14.1. Сводка по элементам проектов MVC

Папка или файл	Описание	Примечания
/App_Data	В эту папку помещаются закрытые данные, такие как XML-файлы или базы данных, если используется SQL Server Express, SQLite или другие хранилища на основе файлов	Веб-сервер IIS не будет обслуживать содержимое этой папки
/App_Start	Эта папка содержит ряд основных настроек конфигурации для проекта, в том числе определение маршрутов и фильтров, а также пакетов содержимого	Маршруты рассматриваются в главах 15 и 16, фильтры — в главе 18, а пакеты содержимого — в главе 26
/Areas	Области — это способ разделения крупного приложения на меньшие части	Области описаны в главе 15
/bin	Сюда помещается скомпилированная сборка приложения MVC вместе со всеми ссылаемыми сборками, находящимися не в GAC	Чтобы увидеть папку bin в окне Solution Explorer (Проводник решения), понадобится щелкнуть на кнопке Show All Files (Показать все файлы). Поскольку все файлы в этой папке являются двоичными и генерируются в результате компиляции, обычно они не хранятся в системе управления исходным кодом
/Content	Сюда помещается статическое содержимое, такое как CSS-файлы и изображения	Это является необязательным соглашением. Статическое содержимое можно хранить в любом подходящем месте
/Controllers	Сюда помещаются классы контроллеров	Это является соглашением. Классы контроллеров могут размещаться где угодно, поскольку они компилируются в ту же самую сборку
/Models	Сюда помещаются классы моделей представлений и моделей предметной области, хотя все кроме простейших приложений выигрывают от определения модели предметной области в отдельном проекте, как было продемонстрировано в приложении SportsStore	Это является соглашением. Классы моделей могут быть определены где угодно в текущем проекте или вообще вынесены в отдельный проект
/Scripts	Эта папка предназначена для хранения библиотек JavaScript, используемых в приложении. По умолчанию Visual Studio добавляет библиотеки jQuery и несколько других популярных JavaScript-библиотек	Это является соглашением. Файлы сценариев могут находиться в любом месте, т.к. в действительности они представляют собой просто другой тип статического содержимого. Дополнительные сведения об управлении файлами сценариев можно найти в главе 26

Окончание табл. 14.1

Папка или файл	Описание	Примечания
/Views	В этой папке хранятся представления и частичные представления, обычно сгруппированные вместе в папках с именами контроллеров, с которыми они связаны	Файл /Views/Web.config предотвращает обслуживание веб-сервером IIS содержимого этих папок. Представления должны визуализироваться через методы действий
/Views/Shared	В этой папке хранятся компоновки и представления, не являющиеся специфичными для какого-либо контроллера	
/Views/Web.config	Это *не* конфигурационный файл для всего приложения. В нем содержится конфигурационная информация, которая обеспечивает обработку представлений с помощью ASP.NET и предотвращает их обслуживание веб-сервером IIS, а также пространства имен, по умолчанию импортируемые в представления	
/Global.asax	Это глобальный класс приложения ASP.NET. В его файле отделенного кода (Global.asax.cs) регистрируется конфигурация маршрутов, а также предоставляется любой код, который должен выполняться при запуске или завершении приложения либо в случае возникновения необработанного исключения	В приложении MVC файл Global.asax играет ту же самую роль, что и в приложении Web Forms
/Web.config	Конфигурационный файл для приложения	В приложении MVC файл Web.config играет ту же самую роль, что и в приложении Web Forms

Соглашения в MVC

В проекте MVC применяются два вида соглашений. Соглашения первого вида — это просто предположения о том, как может выглядеть структура проекта. Например, общепринято размещать файлы JavaScript в папке Scripts. Здесь их рассчитывают обнаружить другие разработчики, использующие MVC, и сюда будут устанавливаться пакеты NuGet. Однако вы вольны переименовать папку Scripts или вообще удалить ее, разместив файлы сценариев где-то в другом месте. Это не помешает инфраструктуре MVC Framework запустить ваше приложение при условии, что элементы script внутри представлений ссылаются на местоположение, в котором находятся файлы сценариев.

Соглашения второго вида произрастают из принципа *соглашения по конфигурации* (convention over configuration; или *соглашения над конфигурацией*, если делать акцент на преимуществе соглашения перед конфигурацией), который был одним из главных аспектов, обеспечивших популярность платформе Ruby on Rails. Соглашение по конфигурации означает, что вы не должны явно конфигурировать, к примеру, ассоциации между контроллерами и их представлениями. Нужно просто следовать определенному соглашению об именовании для файлов — и все будет работать. При соглашении такого рода снижается гибкость в изменении структуры проекта. В последующих разделах объясняются соглашения, которые используются вместо конфигурации.

Совет. Как будет показано в главе 20, все соглашения могут быть изменены в случае использования специального механизма визуализации, однако к такому решению следует подходить серьезно. По большей части в проектах MVC вам придется иметь дело с рассматриваемыми здесь соглашениями.

Следование соглашениям для классов контроллеров

Классы контроллеров должны иметь имена, заканчивающиеся на `Controller`, например, `ProductController`, `AdminController` и `HomeController`. При ссылке на контроллер где-либо в проекте, скажем, при вызове вспомогательного метода HTML, указывается первая часть имени (такая как `Product`), а инфраструктура MVC Framework автоматически добавляет к этому имени слово `Controller` и начинает поиск класса контроллера.

Совет. Это поведение можно изменить, создав собственную реализацию интерфейса `IControllerFactory`, как будет описано в главе 19.

Следование соглашениям для представлений

Представления и частичные представления располагаются в папке `/Views/`*`ИмяКонтроллера`*.

Например, представление, ассоциированное с классом `ProductController`, должно находиться в папке `/Views/Product`.

Совет. Обратите внимание, что часть `Controller` имени класса в имени папки внутри `Views` не указывается, т.е. используется папка `/Views/Product`, *а не* `/Views/ProductController`. Поначалу такой подход может показаться нелогичным, но это очень скоро войдет в привычку.

Инфраструктура MVC Framework ожидает, что стандартное представление для метода действия должно иметь имя этого метода. Например, представление, ассоциированное с методом действия `List()`, должно называться `List.cshtml`. Таким образом, ожидается, что для метода действия `List()` в классе `ProductController` стандартным представлением будет `/Views/Product/List.cshtml`.

Стандартное представление используется при возвращении результата вызова метода `View()` в методе действия, примерно так:

```
return View();
```

Можно указать имя другого представления:

```
return View("MyOtherView");
```

Обратите внимание, что мы не включаем в представление расширение имени файла или путь. При поиске представления MVC Framework просматривает папку, имеющую имя контроллера, и затем папку `/Views/Shared`. Это значит, что представления, применяемые более чем одним контроллером, можно поместить в папку `/Views/Shared` и инфраструктура найдет их самостоятельно.

Следование соглашениям для компоновок

Соглашение об именовании для компоновок предусматривает добавление к имени файла символа подчеркивания (_), а сами файлы компоновки помещаются в папку `/Views/Shared`. Среда Visual Studio создает компоновку по имени `_Layout.cshtml` для всех шаблонов проектов кроме Empty. Эта компоновка по умолчанию применяется ко всем стандартным представлениям через файл `/Views/_ViewStart.cshtml`. Если вы не

хотите, чтобы стандартная компоновка применялась к представлениям, можете изменить настройки в _ViewStart.cshtml (либо вообще удалить этот файл), указав другую компоновку в представлении, например:

```
@{
    Layout = "~/Views/Shared/MyLayout.cshtml";
}
```

Или же можно отключить компоновку для заданного представления:

```
@{
    Layout = null;
}
```

Отладка приложений MVC

Отладка приложения ASP.NET MVC может быть организована тем же способом, что и отладка приложения ASP.NET Web Forms. Отладчик Visual Studio является мощным и гибким инструментом, обладающим множеством средств и возможностей. В этой книге мы рассмотрим его лишь поверхностно, показав, как настраивать отладчик и выполнять различные действия по отладке проекта MVC.

Подготовка проекта для примера

Для демонстрации использования отладчика мы создали новый проект MVC с применением шаблона MVC, просто чтобы вы увидели, как устанавливается стандартное содержимое и конфигурация, а также ознакомились с результатом применения стандартной темы к представлениям. Новый проект назван DebuggingDemo, как показано на рис. 14.3. Был выбран вариант аутентификации Individual User Accounts (Индивидуальные пользовательские учетные записи), который настраивает базовую систему безопасности пользователей.

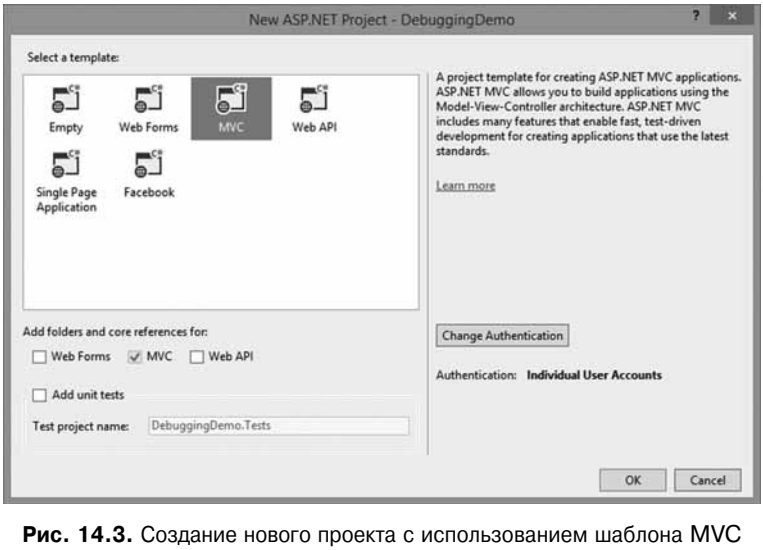

Рис. 14.3. Создание нового проекта с использованием шаблона MVC

Щелкните на кнопке OK. Среда Visual Studio создаст проект и добавит стандартные пакеты, файлы и папки, которые включает шаблон MVC. На рис. 14.4 демонстрируется применение файлов и настроек, добавленных в проект.

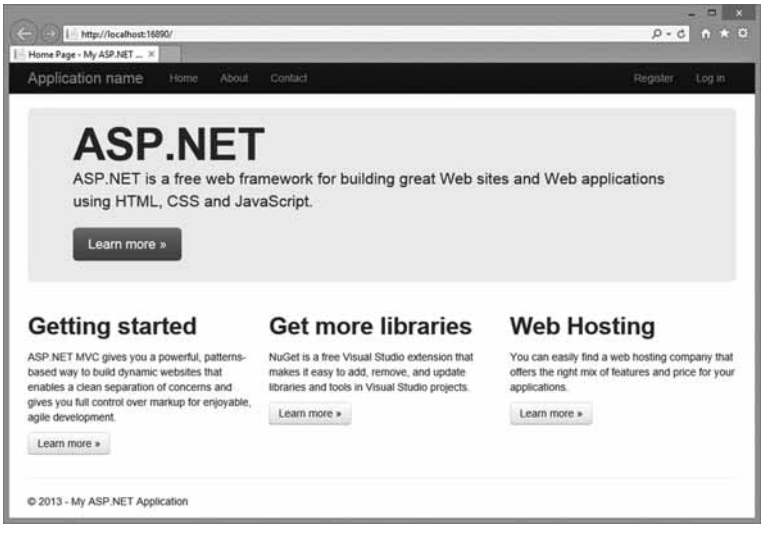

Рис. 14.4. Добавления, сделанные шаблоном проекта MVC

Здесь присутствуют элементы-заполнители для названия приложения и бренда и несколько указателей на документацию по MVC, NuGet и веб-хостинг. В верхней части экрана расположена панель навигации того же рода, что использовалась в приложении SportsStore, а компоновка обладает рядом возможностей чувствительного дизайна. Чтобы увидеть создаваемый ими эффект, измените ширину окна.

Создание контроллера

В качестве части первоначального содержимого проекта среда Visual Studio создала контроллер `Home`, но мы заменяем добавленный ею код тем, который приведен в листинге 14.1.

Листинг 14.1. Содержимое файла `HomeController.cs`

```
using System.Web.Mvc;

namespace DebuggingDemo.Controllers {
    public class HomeController : Controller {

        public ActionResult Index() {
            int firstVal = 10;
            int secondVal = 5;
            int result = firstVal / secondVal;

            ViewBag.Message = "Welcome to ASP.NET MVC!";

            return View(result);
        }
    }
}
```

Создание представления

Среда Visual Studio также создала файл представления `Views/Home/Index.cshtml` как часть настройки проекта. Нас не интересует его стандартное содержимое, поэтому мы заменяем его разметкой, показанной в листинге 14.2.

Листинг 14.2. Содержимое файла `Index.cshtml`

```
@model int

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
    <title>Index</title>
</head>
<body>
    <h2 class="message">@ViewData["Message"]</h2>
    <p>
        The calculation result value is: @Model
    </p>
</body>
</html>
```

Последний подготовительный шаг, который понадобится предпринять — добавить стиль в файл `/Content/Site.css` и изменить один из существующих стилей (листинг 14.3). Файл `Site.css` создан средой Visual Studio в виде части шаблона проекта MVC и является стандартным местом размещения стилей CSS для приложения. (В листинге 14.2 к представлению был добавлен элемент `link`, который импортирует этот файл в `Index.cshtml`.)

Листинг 14.3. Добавление стиля в файл `/Content/Site.css`

```
body { padding-top: 5px; padding-bottom: 5px; }
.field-validation-error { color: #b94a48; }
.field-validation-valid { display: none; }
input.input-validation-error { border: 1px solid #b94a48; }
input[type="checkbox"].input-validation-error { border: 0 none; }
.validation-summary-errors { color: #b94a48; }
.validation-summary-valid { display: none; }
.no-color { background-color: white; border-style:none; }
.message { font-size: 20pt; text-decoration: underline;}
```

Запуск отладчика Visual Studio

Среда Visual Studio подготавливает новые проекты для отладки автоматически, однако полезно разобраться, каким образом можно изменять конфигурацию отладки. Важная настройка, связанная с отладкой, находится в файле `Web.config`, расположенном в корневой папке проекта, внутри элемента `system.web` (листинг 14.4).

Листинг 14.4. Атрибут `debug` в файле `Web.config`

```
...
<system.web>
  <compilation debug="true" targetFramework="4.5.1" />
  <httpRuntime targetFramework="4.5.1" />
</system.web>
...
```

При выполнении приложения на сервере IIS производится много работ по компиляции проекта MVC Framework, поэтому необходимо удостовериться, что во время процесса разработки атрибут `debug` дескриптора `compilation` установлен в `true`. Это гарантирует возможность отладчику взаимодействовать с файлами классов, получаемыми при компиляции по требованию.

Внимание! Не развертывайте приложение на производственном сервере, не установив значение `debug` в `false`. Если для развертывания приложения применяется среда Visual Studio (как это делалось в главе 13), то эта настройка будет изменена автоматически при выборе для проекта конфигурации `Release`.

В дополнение к файлу `Web.config` понадобится обеспечить включение средой Visual Studio отладочной информации в создаваемые ею файлы классов. Это не является критичным, но может вызвать проблемы, если разные настройки отладки не синхронизированы. Удостоверьтесь, что в панели инструментов Visual Studio выбрана конфигурация Debug (Отладочная), как показано на рис. 14.5.

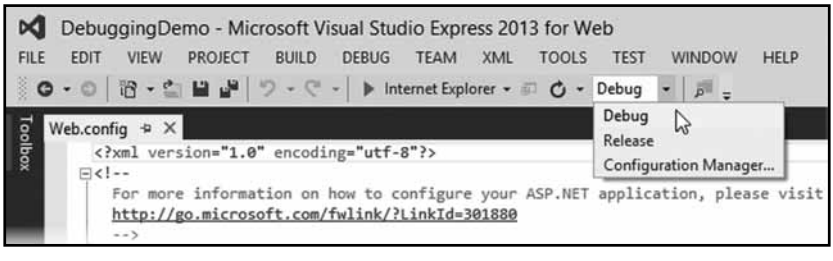

Рис. 14.5. Выбор конфигурации Debug

Чтобы запустить отладку приложения MVC Framework, выберите пункт Start Debugging (Начать отладку) в меню Debug (Отладка) среды Visual Studio или щелкните на стрелке зеленого цвета в панели инструментов Visual Studio (на рис. 14.5 она находится слева от названия браузера, который будет использоваться для отображения приложения — в данном случае это Internet Explorer).

Если при запуске отладчика атрибут `debug` в файле `Web.config` установлен в `false`, среда Visual Studio отобразит диалоговое окно, показанное на рис. 14.6. Выберите переключатель Modify the Web.config file to enable debussing (Модифицировать файл `Web.config` для включения отладки), щелкните на кнопке OK и отладчик запустится.

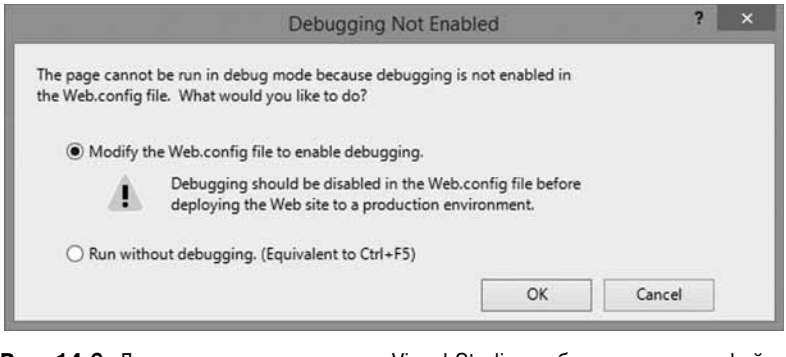

Рис. 14.6. Диалоговое окно, которое Visual Studio отображает, если в файле `Web.config` отладка отключена

В этот момент приложение запустится и отобразится в новом окне браузера, как показано на рис. 14.7. Отладчик присоединится к приложению, но вы не заметите никакой разницы до тех пор, пока не произойдет останов при отладке (что это значит — объясняется в следующем разделе). Чтобы остановить отладчик, выберите пункт Stop Debugging (Остановить отладку) в меню Debug среды Visual Studio или закройте окно браузера.

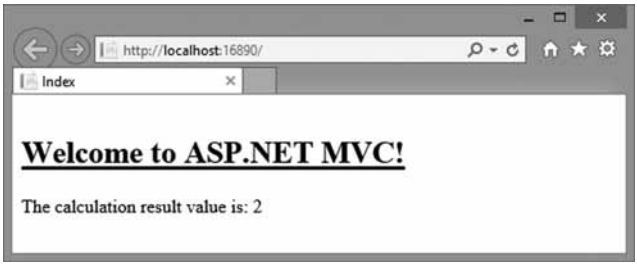

Рис. 14.7. Запуск отладчика

Останов отладчика Visual Studio

Приложение, выполняющееся с присоединенным отладчиком, будет вести себя нормально до тех пор, пока не произойдет *останов*, при котором оно перестает выполняться, а управление передается отладчику. В течение такого останова можно проверять и управлять состоянием приложения. Остановы происходят по двум основным причинам: когда достигнута точка останова и когда произошло необработанное исключение. В последующих разделах будут приведены примеры обеих ситуаций.

Использование точек останова

Точка останова — это инструкция, которая сообщает отладчику о необходимости остановить выполнение приложения и передать управление программисту. После этого можно проинспектировать состояние приложения, посмотреть, что произошло, и (необязательно) возобновить выполнение.

Чтобы создать точку останова, щелкните правой кнопкой мыши на операторе кода и выберите в контекстном меню пункт Breakpoint⇨Insert Breakpoint (Точка останова⇨Вставить точку останова). В качестве демонстрации применим точку останова к первому оператору в методе действия `Index()` контроллера `Home`; на полях редактора кода напротив оператора появится точка красного цвета (рис. 14.8).

Рис. 14.8. Применение точки останова к первому оператору в методе действия `Index()`

Для просмотра точки останова в работе запустите отладчик, выбрав пункт Start Debugging в меню Debug. Приложение будет выполняться вплоть до достижения оператора, к которому была применена точка останова, где отладчик приостановит работу приложения и передаст управление вам. Среда Visual Studio подсвечивает точку, в которой выполнение было остановлено, стрелкой желтого цвета, как показано на рис. 14.9.

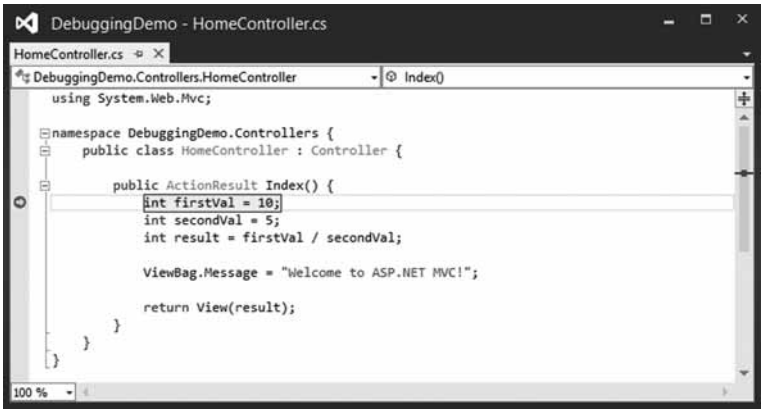

Рис. 14.9. Попадание в точку останова

На заметку! Точка останова активизируется, только когда выполняется ассоциированный с ней оператор. Рассматриваемый пример точки останова достигается сразу после запуска приложения, поскольку она находится внутри метода действия, который вызывается при получении запроса к стандартному URL. Если поместить точку останова внутрь другого метода действия, понадобится с помощью браузера запросить URL, ассоциированный с этим методом. Это может означать работу с приложением так, как это делает пользователь, или навигацию непосредственно к URL в окне браузера.

Получив контроль над выполнением приложения, вы можете перейти на следующий оператор, зайти внутрь других методов и в целом исследовать состояние приложения. Все это можно делать с помощью кнопок панели инструментов или пунктов меню Debug среды Visual Studio. Вдобавок к предоставлению контроля над выполнением приложения среда Visual Studio предлагает множество полезной информации о состоянии приложения; в действительности информации настолько много, что в этой книге удастся рассмотреть только самые базовые сведения.

Просмотр значений данных в редакторе кода

Наиболее распространенное использование точек останова связано с отслеживанием ошибок в коде. Перед тем как можно будет исправить ошибку, понадобится выяснить, что происходит, и одним самых полезных средств, предлагаемых Visual Studio, является возможность просмотра и отслеживания значений переменных прямо в редакторе кода.

В качестве примера запустите приложение с применением отладчика и подождите, пока не будет достигнута точка останова, добавленная в предыдущем разделе. Когда отладчик остановится, переместите курсор мыши на оператор, в котором определяется переменная `result`. Вы увидите небольшое всплывающее окно, отображающее текущее значение переменной, как показано на рис. 14.10 (для лучшего восприятия здесь также приведена и увеличенная версия этого окна).

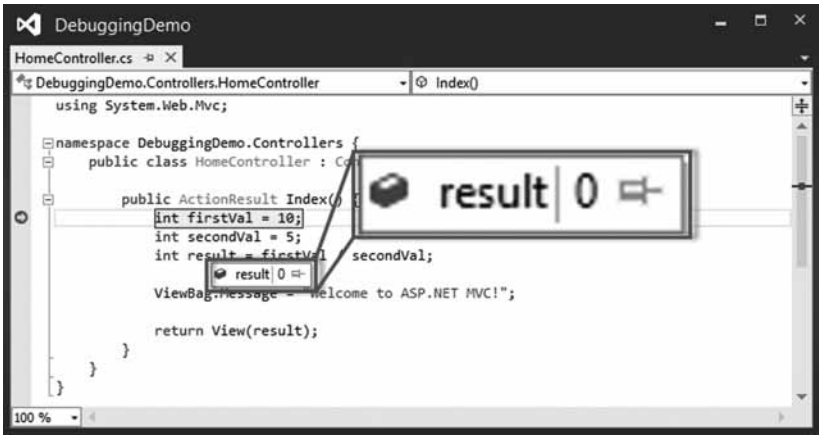

Рис. 14.10. Отображение значения переменной в редакторе кода Visual Studio

Выполнение операторов в методе действия `Index()` не достигло точки, где переменной `result` присваивается значение, поэтому Visual Studio показывает стандартное значение, равное `0` для типа `int`. Выберите в меню Debug пункт Step Over (Выполнить без захода) или нажмите <F10>, чтобы переместить точку выполнения к оператору, который определяет свойство `ViewBar.Message`, и снова наведите курсор мыши на переменную `result`. Мы выполнили оператор, присваивающий значение переменной `result`, и эффект можно видеть на рис. 14.11.

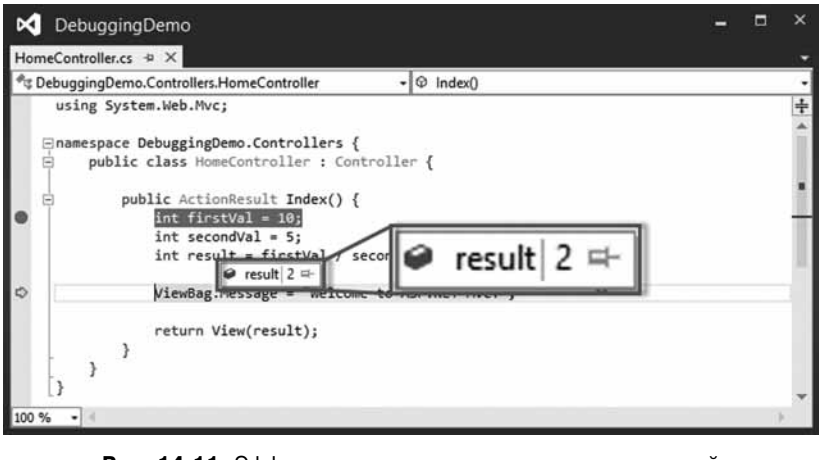

Рис. 14.11. Эффект от присваивания значения переменной

Описанное средство применяется в начале процесса отслеживания ошибки, поскольку он позволяет немедленно понять, что происходит внутри приложения; оно особенно полезно при выявлении значений `null`, которые указывают на то, что переменной не было присвоено значение (распространенная причина многих ошибок).

Вы наверняка заметили во всплывающем окне значок с изображением канцелярской кнопки справа от значения. Щелчок на нем приводит к тому, что всплывающее окно становится постоянным, и будет отражать изменения значения переменной — это позволяет отслеживать одну или более переменных и видеть, когда они изменяются и каковы их новые значения.

Просмотр состояния приложения в окне отладчика

Среда Visual Studio предоставляет несколько разных окон, которые можно использовать для получения информации о приложении, пока выполнение приостановлено в точке останова. Полный список доступных окон находится в меню Debug⇨Windows (Отладка⇨Окна), а двумя наиболее часто применяемыми окнами являются Locals (Локальные) и Call Stack (Стек вызовов). Окно Locals автоматически отображает значения всех переменных в текущей области видимости, как показано на рис. 14.12. Это обеспечивает всеобъемлющее представление переменных, которое наверняка будет интересным разработчику.

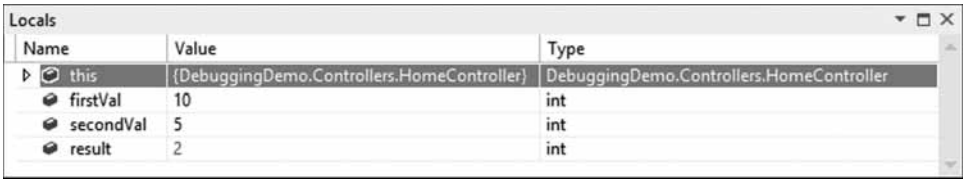

Рис. 14.12. Окно Locals

Переменные, значения которых изменились предыдущим выполненным оператором, показаны красным цветом. На рис 14.12 это переменная `result`, т.к. только что был выполнен оператор, присваивающий ей значение.

Совет. Набор переменных, отображаемый в окне Locals, изменяется по мере прохождения по приложению, но если необходимо видеть какую-то переменную постоянно, следует щелкнуть правой кнопкой мыши на соответствующем элементе в окне Locals и выбрать в контекстном меню пункт Add Watch (Добавить наблюдение). Элементы в окне Watch (Наблюдение) не изменяются по мере выполнения операторов в приложении, предоставляя фиксированную контрольную точку.

Окно Call Stack отображает последовательность вызовов, которая привела к выполнению текущего оператора. Это оказывается очень полезным при попытке выявления причин странного поведения, поскольку можно раскрутить стек вызовов и исследовать обстоятельства, приведшие к активизации точки останова. (Мы не приводим здесь экранный снимок с окном Call Stack, поскольку нашему простому приложению не хватает глубины вызовов, чтобы предоставить сколько-нибудь полезную информацию в этом окне. Однако мы рекомендуем исследовать как упомянутое, так и другие окна Visual Studio, чтобы иметь понятие о том, какие сведения способен предоставлять отладчик.)

Совет. Точки останова можно добавлять и в представления. Например, они очень полезны при анализе значений свойств, относящихся к модели представления. Добавление точки останова в представление осуществляется точно так же, как в файл кода: щелкните правой кнопкой мыши на интересующем операторе Razor и выберите в контекстном меню пункт Breakpoint⇨Insert Breakpoint (Точка останова⇨Вставить точку останова).

Остановы из-за исключений

Необработанные исключения — это явление, присущее процессу разработки. Одна из причин проведения модульного и интеграционного тестирования в проектах — сведение к минимуму вероятности возникновения таких исключений в производственных версиях. В качестве помощи в поиске и устранении причин возникновения необработанных исключений, отладчик Visual Studio будет автоматически останавливаться, когда встретит такое исключение.

На заметку! К останову отладчика приводят только *необработанные* исключения. Исключение становится *обработанным*, если оно было перехвачено и обработано в блоке `try...catch`. Обработанные исключения представляют собой полезный инструмент в программировании. Они используются для моделирования сценария, при котором метод не может завершить свою задачу и должен уведомить об этом вызывающий код. Необработанные исключения нежелательны, т.к. они представляют *непредвиденное* условие и в общем случае переносят пользователя на страницу ошибки.

Для демонстрации останова по исключению мы внесли небольшое изменение в метод действия `Index()` контроллера `Home`, как показано в листинге 14.5.

Листинг 14.5. Добавление оператора, который вызовет исключение, в файле `HomeController.cs`

```
using System.Web.Mvc;
namespace DebuggingDemo.Controllers {
   public class HomeController : Controller {
      public ActionResult Index() {
         int firstVal = 10;
         int secondVal = 0;
         int result = firstVal / secondVal;
         ViewBag.Message = "Welcome to ASP.NET MVC!";
         return View(result);
      }
   }
}
```

Мы изменили значение переменной `secondVal` на `0`, что приведет к генерации исключения в операторе, в котором производится деление `firstVal` на `secondVal`.

На заметку! Мы также удалили точку останова из метода действия `Index()`, для чего щелкнули правой кнопкой мыши на значке точки останова, расположенном на полях редактора кода, и выбрали в контекстном меню пункт Delete Breakpoint (Удалить точку останова).

После запуска отладчика приложение будет выполняться вплоть до генерации исключения. Затем отобразится вспомогательное всплывающее окно исключения, показанное на рис. 14.13. Вспомогательное всплывающее окно исключения предоставляет подробности, связанные с исключением. Когда отладчик останавливается по исключению, можно проинспектировать состояние приложения и потока управления, как это обычно делается при достижении обычной точки останова.

Использование средства Edit and Continue

Одно из интересных средств отладки Visual Studio называется Edit and Continue (Отредактировать и продолжить). Когда отладчик останавливается, вы можете отредактировать код и продолжить отладку. Среда Visual Studio перекомпилирует приложение и воссоздаст состояние приложения на момент останова отладки.

Включение средства Edit and Continue

Средство Edit and Continue должно быть включено в двух местах.

- В разделе Edit and Continue (Отредактировать и продолжить) параметров Debugging (Отладка) диалогового окна Options (Параметры) (для его открытия выберите пункт Options в меню Tools (Сервис) среды Visual Studio) удостоверьтесь, что флажок Enable Edit and Continue (Включить средство Edit and Continue) отмечен, как показано на рис. 14.14.

- В окне свойств проекта (для его открытия выберите пункт DebuggingDemo Properties (Свойства DebuggingDemo) в меню Project (Проект) среды Visual Studio) щелкните на разделе Web (Веб) и удостоверьтесь, что флажок Enable Edit and Continue отмечен, как показано на рис. 14.15.

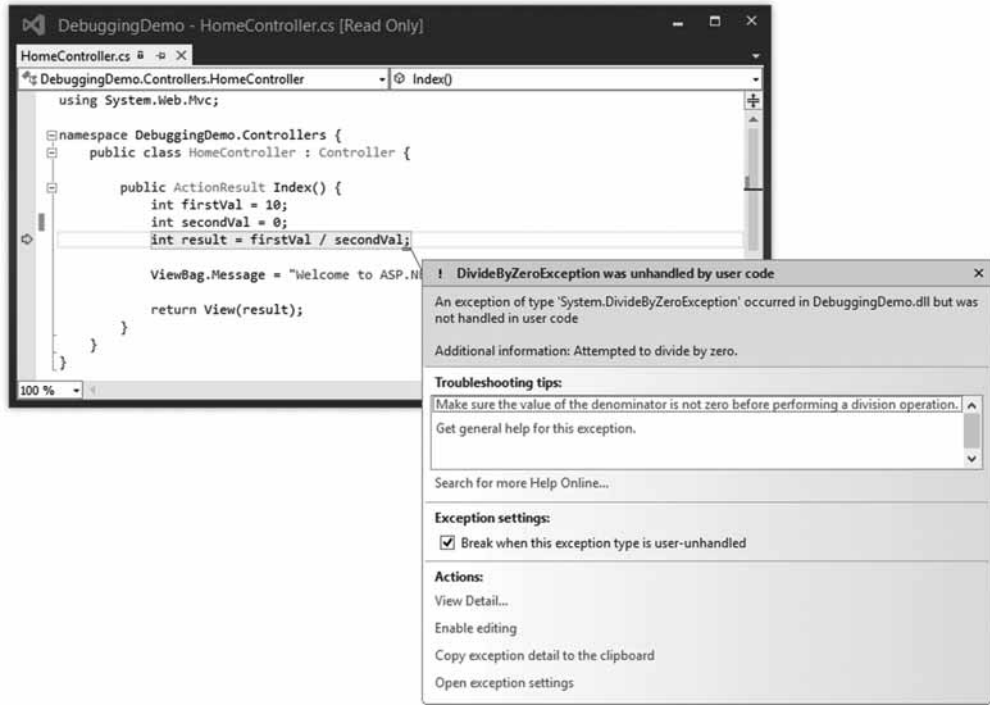

Рис. 14.13. Вспомогательное всплывающее окно исключения

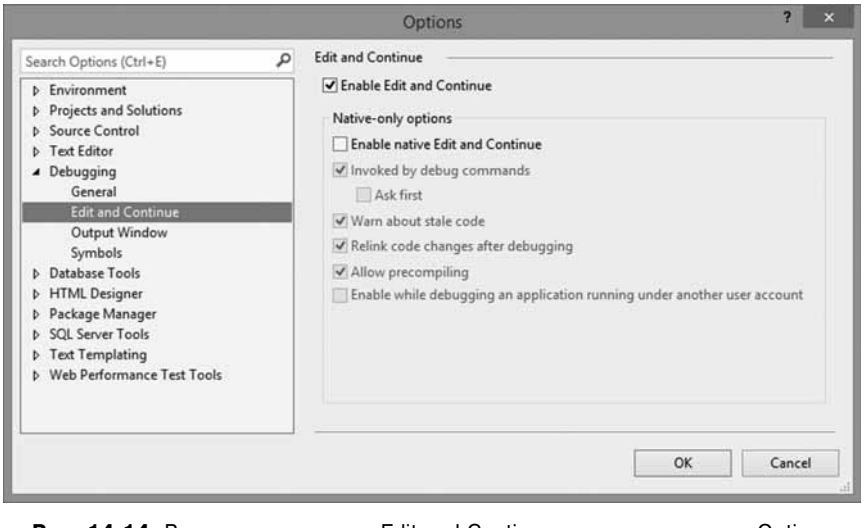

Рис. 14.14. Включение средства Edit and Continue в диалоговом окне Options

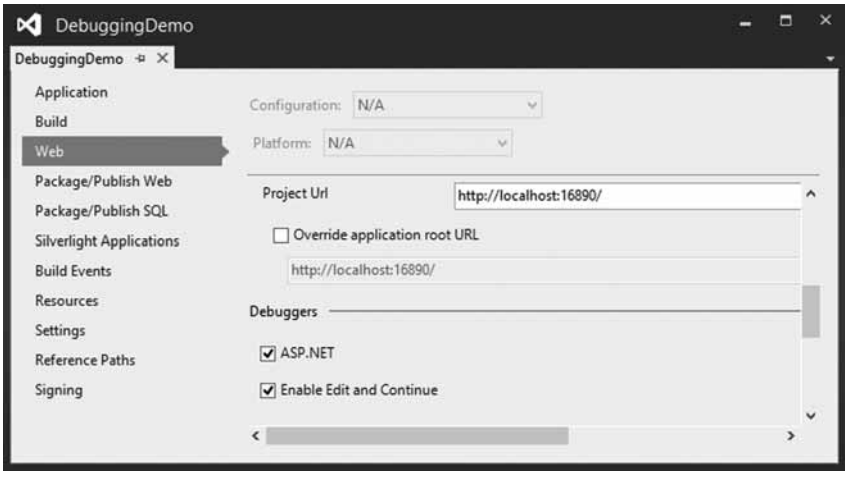

Рис. 14.15. Включение средства Edit and Continue в окне свойств проекта

Внесение модификаций в проект

Со средством Edit and Continue связаны кое-какие требования. Существуют условия, при которых оно работать не будет. Одно из таких условий присутствует в методе действия `Index()` класса `HomeController`: использование динамических объектов. Чтобы обойти это, мы закомментируем строку кода в классе `HomeController.cs`, в которой применяется объект `ViewBag`, как показано в листинге 14.6.

Листинг 14.6. Удаление обращения к `ViewBag` из метода `Index()` в файле `HomeController.cs`

```
using System.Web.Mvc;

namespace DebuggingDemo.Controllers {
    public class HomeController : Controller {

        public ActionResult Index() {
            int firstVal = 10;
            int secondVal = 0;
            int result = firstVal / secondVal;

            // Этот оператор закомментирован
            //ViewBag.Message = "Welcome to ASP.NET MVC!";

            return View(result);
        }
    }
}
```

В представление `Index.cshtml` понадобится внести соответствующее изменение (листинг 14.7).

Листинг 14.7. Удаление обращения к `ViewBag` из файла `Index.cshtml`

```
@model int

@{
    Layout = null;
}
```

```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link href="~/Content/Site.css" rel="stylesheet" type="text/css" />
    <title>Index</title>
</head>
<body>
    <!-- Этот элемент закомментирован -->
    <!--<h2 class="message">@ViewData["Message"]</h2>-->
    <p>
        The calculation result value is: @Model
    </p>
</body>
</html>
```

Демонстрация работы средства Edit and Continue

Теперь все готово для демонстрации работы средства Edit and Continue. Начните с выбора пункта Start Debugging в меню Debug. Приложение запустится с присоединенным отладчиком, и будет выполняться вплоть до достижения строки в методе `Index()`, где производится простое вычисление. Значением второго параметра является `0`, поэтому генерируется исключение. В этой точке отладчик останавливает выполнение и отображается вспомогательное всплывающее окно исключения (подобное показанному на рис. 14.13).

Щелкните на ссылке Enable editing (Включить редактирование) во вспомогательном всплывающем окне исключения. В редакторе кода измените выражение, которое вычисляет значение переменной `result`, следующим образом:

```
...
int result = firstVal / 2;
...
```

Ссылка на переменную `secondVal` была удалена, а вместо нее введено числовое литеральное значение 2. Теперь выберите пункт Continue (Продолжить) в меню Debug среды Visual Studio, чтобы возобновить выполнение приложения. Новое значение используется для получения значения для переменной `result`, давая вывод, представленный на рис. 14.16.

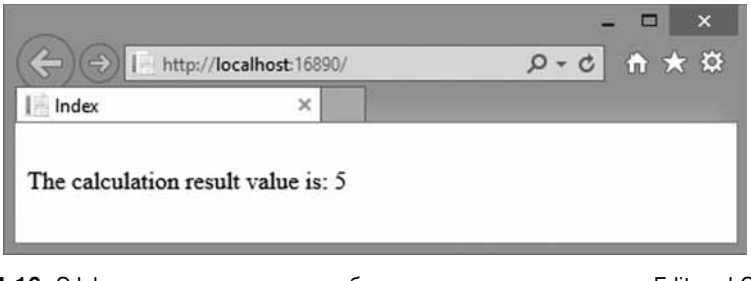

Рис. 14.16. Эффект от исправления ошибки с применением средства Edit and Continue

Потратьте некоторое время на обдумывание того, что здесь произошло. Мы запустили приложение, содержащее ошибку — попытку деления на ноль. Отладчик обнаружил исключение и остановил выполнение программы. Мы отредактировали код и затем сообщили отладчику о необходимости продолжить выполнение. В этой точке среда Visual Studio перекомпилировала приложение, воссоздала состояния и продолжила выполне-

ние обычным образом, используя новое значение. Браузер получает визуализированный результат, который отражает это новое значение. Без средства Edit and Continue пришлось бы остановить приложение, внести изменение, скомпилировать приложение и перезапустить отладчик. Затем понадобилось бы повторить в браузере все шаги, которые были проделаны до момента останова отладчика. Наиболее важным может оказаться именно устранение этой необходимости в последнем действии. Воспроизведение сложных ошибок часто требует проведения множества шагов, поэтому возможность тестирования потенциальных исправлений без постоянного повторения таких шагов экономит время и усилия программиста.

Использование ссылок на браузеры

Версия среды Visual Studio 2013 включает средство под названием *ссылок на браузеры*, которое позволяет просматривать приложение одновременно в нескольких браузерах и инициировать его перезагрузку во всех браузерах после внесения изменений. Данное средство наиболее удобно на этапе, когда функциональность приложения стала устойчивой и производятся работы по доводке HTML-разметки и стилей CSS, которые генерируют представления. (Причины вскоре будут объяснены.)

Чтобы воспользоваться ссылкой на браузер, щелкните на небольшом значке с изображением стрелки вниз рядом с выбранным браузером в панели инструментов Visual Studio и выберите пункт Browse With (Просматривать с помощью) в раскрывшемся меню (рис. 14.17).

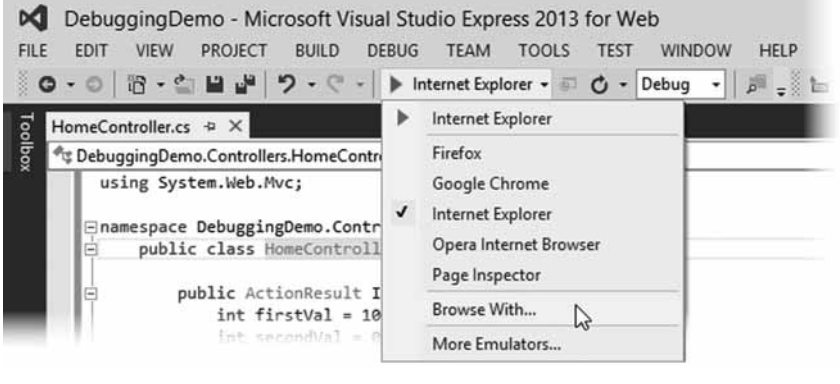

Рис. 14.17. Подготовка к выбору браузеров, применяемых средством ссылок на браузеры

Откроется диалоговое окно Browse With (Просмотр с помощью). Удерживая нажатой клавишу <Ctrl>, выберите необходимые браузеры. На рис. 14.18 можно видеть, что выбраны браузеры Internet Explorer и Chrome. Это диалоговое окно можно также применять для добавления новых браузеров (хотя среда Visual Studio довольно неплохо обнаруживает большинство популярных браузеров).

Щелкните на кнопке Browse (Просмотр). Среда Visual Studio откроет выбранные браузеры и обеспечит загрузку в них URL проекта. Вы можете отредактировать код и представления в приложении и затем обновить все окна браузеров, выбрав пункт Refresh Linked Browsers (Обновить связанные браузеры) в панели инструментов Visual Studio, как показано на рис. 14.19. Приложение автоматически скомпилируется, поэтому вы увидите результаты внесенных изменений.

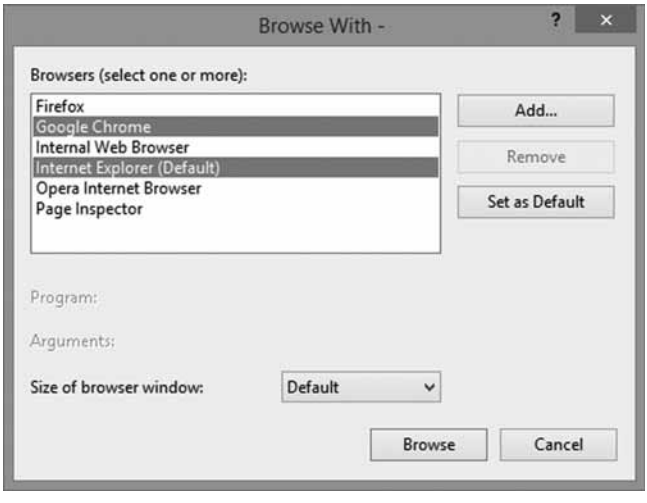

Рис. 14.18. Выбор нескольких браузеров

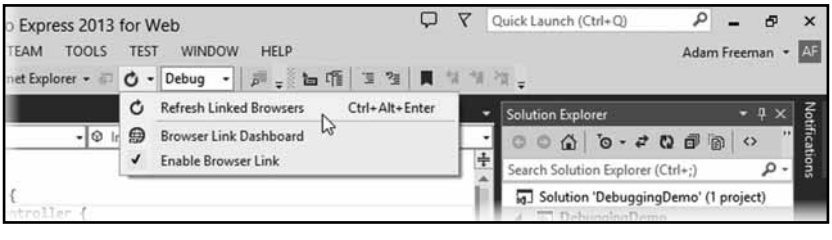

Рис. 14.19. Обновление связанных браузеров

Это средство работает за счет включения определенного кода JavaScript в HTML-разметку, отправляемую браузеру, и может быть удобным способом итеративной разработки. Причина, по которой я рекомендую его только для работы над представлениями, связана с тем, что они реже вызывают отправку веб-сервером IIS сообщений об ошибках HTTP браузеру, возникающих из-за наличия ошибок в коде. Код JavaScript не добавляется в ответы, генерируемые в случае ошибок, а это значит, что связь между средой Visual Studio и браузерами теряется. Придется снова начинать с использования пункта меню Browse With. Средство ссылок на браузеры само по себе является хорошим, но применение кода JavaScript создает проблему. При разработке приложений, отличных от ASP.NET, я использую похожий инструмент под названием LiveReload (`http://livereload.com`), обеспечивающий более удачный подход, поскольку он предполагает применение подключаемых модулей браузеров, на которые сообщения об ошибках HTTP влияния не оказывают. Ценность средства ссылок на браузеры среды Visual Studio будет ограниченной до тех пор, пока в Microsoft не предпримут аналогичный подход.

Резюме

В этой главе была описана структура проекта MVC в Visual Studio и согласование его частей друг с другом. Мы также коснулись одной из самых важных характеристик MVC Framework, которой являются соглашения. По мере исследования внутреннего функционирования инфраструктуры MVC Framework в последующих главах мы будем возвращаться к этим темам снова и снова.