ПРИЛОЖЕНИЕ B

# Введение в ASP.NET Web Forms

Все рассмотренные до сих пор примеры были консольными и настольными приложениями с графическим пользовательским интерфейсом, созданными с применением инфраструктуры WPF. В главах 29 и 30 раскрывались инфраструктуры ASP.NET MVC и ASP.NET Web API, а в этом и последующих двух приложениях — инфраструктура ASP.NET Web Forms. Для начала будет дан краткий обзор ряда основных концепций веб-разработки (HTTP, HTML, сценарии клиентской стороны, обратные отправки) и описана роль коммерческого веб-сервера Microsoft (IIS), а также IIS Express.

После краткого введения в оставшемся материале приложения внимание будет сосредоточено на структуре модели программирования с применением ASP.NET Web Forms (включая однофайловую модель и модель отделенного кода) и функциональности базового класса `Page`. Попутно вы ознакомитесь с ролью веб-элементов управления ASP.NET, структурой каталогов веб-сайта ASP.NET, а также использованием файла `Web.config` для управления оперированием веб-сайтов.

## Роль протокола HTTP

Веб-приложения значительно отличаются от настольных графических приложений. Первая очевидная разница связана с тем, что веб-приложение производственного уровня всегда вовлекает, по меньшей мере, две подключенные к сети машины: на одной располагается веб-сайт, а на другой просматриваются данные внутри веб-браузера. Разумеется, во время разработки вполне возможно, что единственная машина будет исполнять роль и клиента на основе браузера, и веб-сервера, который обслуживает содержимое веб-сайта. Учитывая природу веб-приложений, подключенные к сети машины должны согласовать конкретный сетевой протокол для определения способа отправки и получения данных. Сетевым протоколом, который соединяет такие машины, является HTTP (Hypertext Transfer Protocol — протокол передачи гипертекста).

### Цикл “запрос/ответ” HTTP

Когда на клиентской машине запускается веб-браузер (такой как Google Chrome, Opera, Mozilla Firefox, Apple Safari или Microsoft Internet Explorer/Edge), выполняется *HTTP-запрос* для доступа к определенному ресурсу (обычно веб-странице) на удаленной серверной машине. По существу HTTP представляет собой текстовый протокол на основе стандартной парадигмы “запрос/ответ”. Например, в случае перехода на `www.facebook.com` программное обеспечение браузера задействует веб-технологию под названием *служба доменных имен* (Domain Name Service — DNS), которая преобразует за-

регистрированный URL-адрес в числовое значение, называемое *IP-адресом*. После этого браузер открывает сокетное подключение (обычно через порт 80 для незащищенных подключений) и отправляет HTTP-запрос для обработки на целевом сайте.

Веб-сервер получает входящий HTTP-запрос и может обработать любые отправленные клиентом входные значения (вроде значения внутри текстового поля, состояния флажка или переключателя), чтобы скомпоновать подходящий *HTTP-ответ*. Программисты веб-приложений могут применять любое количество серверных технологий (PHP, ASP.NET, JSP и т.д.) для динамической генерации содержимого, подлежащего встраиванию в HTTP-ответ. Затем браузер клиентской стороны визуализирует HTML-разметку, отправленную веб-сервером. На рис. В.1 показан базовый цикл "запрос/ответ" HTTP.

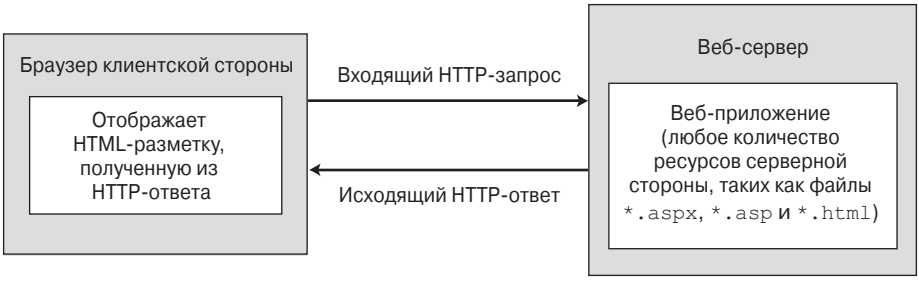



**Рис. В.1.** Цикл "запрос/ответ" HTTP

## HTTP является протоколом без хранения состояния

Еще один аспект веб-разработки, который заметно отличает ее от программирования традиционных настольных приложений, связан с тем фактом, что HTTP на самом деле является сетевым протоколом *без хранения состояния*. Как только веб-сервер отправит ответ клиентскому браузеру, предыдущее взаимодействие забывается. В традиционном настольном приложении, безусловно, все не так: состояние исполняемой программы почти всегда активно до тех пор, пока пользователь не закроет главное окно приложения.

С учетом данного факта на разработчика веб-приложений возлагается ответственность за реализацию специфических действий по "запоминанию" важной информации (такой как элементы корзины покупок, номера кредитных карт и домашние адреса), связанной с пользователями, которые в текущий момент находятся на сайте. Как будет показано в приложении Д, технология ASP.NET Web Forms предлагает множество способов поддержки состояния: переменные сеанса, cookie-наборы и кеш приложения, а также API-интерфейс управления профилями ASP.NET Web Forms.

# Веб-приложения и веб-серверы

*Веб-приложение* можно воспринимать как коллекцию файлов (например, `*.html`, `*.aspx`, файлов изображений, файлов данных XML) и связанных компонентов (таких как библиотека кода .NET), хранящихся в определенном наборе каталогов на веб-сервере. В приложении Д вы увидите, что приложения Web Forms обладают специфическим жизненным циклом и предоставляют многочисленные события (наподобие начального запуска или финального останова), которые можно перехватывать для выполнения специализированной обработки во время функционирования веб-сайта.

*Веб-сервер* — это программный продукт, отвечающий за размещение веб-приложений. Он обычно предлагает множество взаимосвязанных служб, таких как интегрированная система безопасности, поддержка FTP (File Transfer Protocol — протокол передачи файлов),

службы обмена почтой и т.д. Информационные службы Интернета (Internet Information Services — IIS) представляют собой веб-серверный продукт производственного уровня от Microsoft, который обладает встроенной поддержкой приложений Web Forms.

При наличии на рабочей станции корректно установленного сервера IIS с ним можно взаимодействовать через папку Administrative Tools (Администрирование) панели управления, дважды щелкнув на значке Internet Information Services Manager (Диспетчер служб IIS). На рис. B.2 показан узел Default Web Site (Веб-сайт по умолчанию) в IIS, где находится большинство параметров конфигурации (в предшествующих версиях IIS пользовательский интерфейс будет выглядеть по-другому).


**Рис. B.2.** Диспетчер служб IIS позволяет конфигурировать поведение Microsoft IIS во время выполнения

## Роль виртуальных каталогов IIS

Единственная установленная копия IIS способна размещать многочисленные веб-приложения, каждое из которых располагается в своем *виртуальном каталоге*. Каждый виртуальный каталог сопоставлен с физическим каталогом на жестком диске машины. Например, если создается новый виртуальный каталог по имени `CarsAreUs`, то извне на сайт можно перейти с использованием URL вида `www.MyDomain.com/CarsAreUs` (предполагая, что IP-адрес `www.MyDomain.com` был зарегистрирован в DNS). "За кулисами" виртуальный каталог отображается на физический корневой каталог, в котором находится содержимое веб-приложения `CarsAreUs`.

Как будет показано далее в приложении, при создании приложений Web Forms с применением Visual Studio можно заставить IDE-среду автоматически создавать новый виртуальный каталог для текущего веб-сайта. Однако при необходимости создать виртуальный каталог можно и вручную, щелкнув правой кнопкой мыши на узле Default Web Site в диспетчере служб IIS и выбрав в контекстном меню пункт Add Virtual Directory (Добавить виртуальный каталог).

## Веб-сервер IIS Express

В ранних версиях платформы .NET разработчики ASP.NET обязаны были использовать виртуальные каталоги IIS во время построения и тестирования своих веб-приложений. Во многих случаях такая тесная зависимость от IIS излишне усложняла коллективную разработку, не говоря уже о том, что многие сетевые администраторы неодобрительно относились к установке IIS на машине каждого разработчика.

К счастью, теперь появился вариант в виде легковесного веб-сервера под названием IIS Express. Данная утилита позволяет разработчикам размещать приложения Web Forms за границами IIS. С применением IIS Express веб-страницы можно строить и тестировать в любом каталоге на машине. Это удобно в сценариях коллективной разработки, а также при построении приложений Web Forms на машинах с версиями Windows, которые не поддерживают установку IIS.

В большинстве примеров настоящей книги вместо размещения веб-содержимого в виртуальном каталоге IIS используется веб-сервер IIS Express (через соответствующий тип проекта Visual Studio). В то время как такой подход может упростить разработку веб-приложения, имейте в виду, что веб-сервер IIS Express *не* рассчитан на размещение веб-приложений производственного уровня. Он предназначен только для целей разработки и тестирования. Когда веб-приложение готово к эксплуатации, сайт понадобится скопировать в виртуальный каталог IIS.

---

**На заметку!** Среда Visual Studio предлагает встроенный инструмент для копирования локального веб-приложения на производственный веб-сервер. Работа сводится к паре щелчков на кнопках. Чтобы запустить процесс копирования, необходимо выбрать веб-проект в окне Solution Explorer, щелкнуть правой кнопкой мыши и выбрать в контекстном меню пункт Publish (Опубликовать). Затем можно указать желаемое местоположение для развертывания, в том числе Microsoft Azure.

---

# Роль языка HTML

Теперь, когда сконфигурирован каталог для размещения веб-приложения и выбран веб-сервер, который будет служить хостом, понадобится создать само содержимое. Вспомните, что веб-приложение — это просто набор файлов, составляющих функциональность сайта. В действительности многие файлы в наборе будут содержать операторы HTML (Hypertext Markup Language — язык разметки гипертекста). Важно знать, что HTML является стандартным языком разметки для описания того, как литеральный текст, изображения, внешние ссылки и разнообразные элементы управления HTML должны визуализироваться внутри браузера клиентской стороны.

Хотя современные IDE-среды (включая Visual Studio) и платформы для разработки веб-приложений (такие как ASP.NET) генерируют большой объем HTML-разметки автоматически, при работе с ASP.NET вам потребуются практические знания языка HTML.

---

**На заметку!** Вспомните из главы 2, что компания Microsoft выпустила несколько бесплатных IDE-сред в рамках семейства продуктов Express, а также версию Visual Studio Community, которая объединяет все редакции Express в один пакет. Для проработки материала последующих приложений, посвященных Web Forms, можно загрузить редакцию Visual Studio Express для Web или Visual Studio Community.

---

В этом разделе рассматриваются основы HTML, которые способствуют пониманию разметки, генерируемой моделью программирования Web Forms.

## Структура документа HTML

Типичный файл HTML состоит из набора дескрипторов, описывающих внешний вид и поведение веб-страницы. Базовая структура документа HTML имеет тенденцию оставаться той же самой. Например, файлы `*.html` начинаются и завершаются с помощью дескрипторов `<html>` и `</html>`, обычно имеют раздел `<body>` и т.д.

Начнем с открытия IDE-среды Visual Studio и выбора пункта меню New⇨Project (Создать⇨Проект). В узле Other Project Types (Другие типы проектов) выберем элемент Visual Studio Solutions (Решения Visual Studio) и затем вариант Blank Solution (Пустое решение), как показано на рис. В.3 (обратите внимание, что веб-проект сейчас не строится, а просто создается пустое решение для хранения файлов).

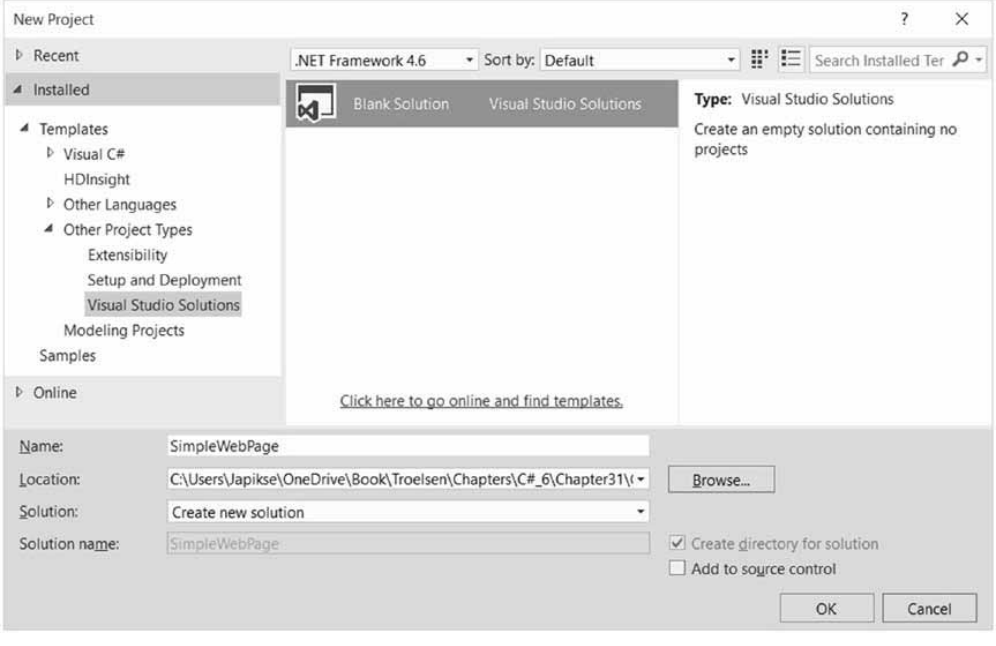



**Рис. В.3.** Выбор варианта Blank Solution в диалоговом окне создания проекта

Добавим пустую страницу HTML, выбрав пункт меню Project⇨Add New Item (Проект⇨Добавить новый элемент) и указав Visual C#/Web в панели слева и HTML Page (Страница HTML) в панели по центру. Назначим файлу имя `HtmlPage1.html`. Должна отобразиться следующая начальная разметка (в зависимости от конфигурации Visual Studio она может отличаться):

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title> </title>
  <meta charset="utf-8" />
</head>
<body>

</body>
</html>
```

Прежде всего, обратите внимание, что данный файл HTML начинается с инструкции обработки DOCTYPE. В сочетании с открывающим дескриптором <html> это указывает, что содержащиеся в файле дескрипторы HTML должны проверяться на соответствие стандарту HTML 5.0. Стандарт HTML 5.0 представляет собой спецификацию W3C, которая добавляет к базовому языку разметки много новых возможностей.

---

**На заметку!** По умолчанию Visual Studio проверяет все документы HTML на соответствие схеме HTML 5.0, чтобы обеспечить согласованность разметки со стандартом HTML 5. Если нужно указать альтернативную схему проверки достоверности, тогда понадобится открыть диалоговое окно Options (Параметры), выбрав пункта меню Tools⇨Options (Сервис⇨Параметры), развернуть узел Text Editor (Текстовый редактор), затем узел HTML (Web Forms) и выбрать узел Validation (Проверка достоверности). Если видеть предупреждения проверки нежелательно, то следует просто снять отметку с расположенного там же флажка Show Errors (Показывать ошибки).

---

Чтобы немного приукрасить начальную страницу, изменим ее заголовок:

```
<head>
  <title>This is my simple web page</title>
</head>
```

Дескрипторы <title> вполне ожидаемо применяются для указания текстовой строки, которая должна быть помещена в область заголовка окна веб-браузера.

## Роль формы HTML

*Форма HTML* — это просто именованная группа связанных элементов пользовательского интерфейса, обычно используемых для сбора пользовательского ввода. Не путайте форму HTML с полной областью отображения заданного браузера. В действительности форма HTML в большей степени представляет собой *логическую группу* графических элементов управления, помещенных между дескрипторами <form> и </form>, например:

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>This is my simple web page</title>
</head>
<body>
  <form id="defaultPage">
    <!-- Вставить сюда содержимое пользовательского веб-интерфейса -->
  </form>
</body>
</html>
```

Форме был назначен идентификатор defaultPage. Обычно открывающий дескриптор <form> содержит атрибут action, который указывает URL-адрес, применяемый для отправки данных формы, а также атрибут method, определяющий метод передачи данных (POST или GET). Вы узнаете о них в следующем разделе. А пока давайте взглянем на виды элементов, которые могут быть помещены в форму HTML (помимо обычного текста).

## Инструменты визуального конструктора разметки HTML в Visual Studio

В панели инструментов (доступной через пункт меню View⇨Toolbox (Вид⇨Панель инструментов)) среды Visual Studio есть вкладка HTML, которая позволяет выбрать элемент управления HTML и поместить его на поверхность визуального конструктора разметки HTML (рис. B.4). Аналогично процессу построения приложения WPF элементы управления можно перетаскивать на поверхность конструктора или прямо в разметку страницы.

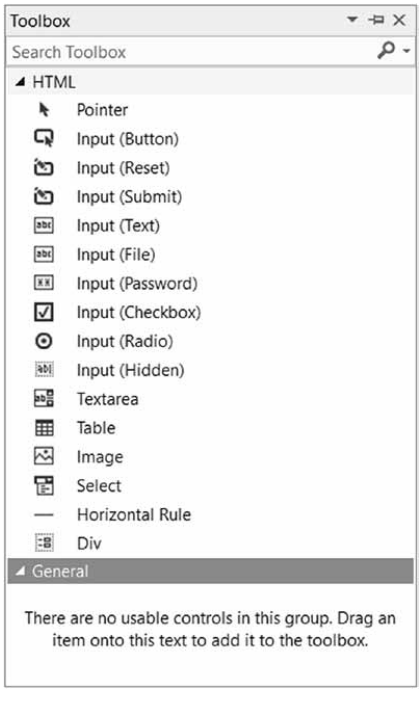



**Рис. B.4.** Вкладка HTML панели инструментов

**На заметку!** При построении веб-страниц ASP.NET с использованием модели программирования Web Forms такие элементы управления HTML обычно не применяются для создания пользовательского интерфейса. Вместо них используются элементы управления Web Forms, которые будут преобразованы в корректную разметку HTML без вашего участия. Роль веб-элементов управления обсуждается далее в приложении.

Редактор HTML не имеет поверхности визуального конструктора. Для работы с визуальным конструктором (или применения режима разделения) понадобится использовать редактор HTML из Web Forms. Чтобы сделать это, закроем окно редактора для `HtmlPage1.html` и в окне Solution Explorer щелкнем правой кнопкой мыши на имени файла, выберем в контекстном меню пункт Open With (Открыть с помощью), в результате чего откроется диалоговое окно, которое позволяет указать необходимый редактор (рис. B.5).

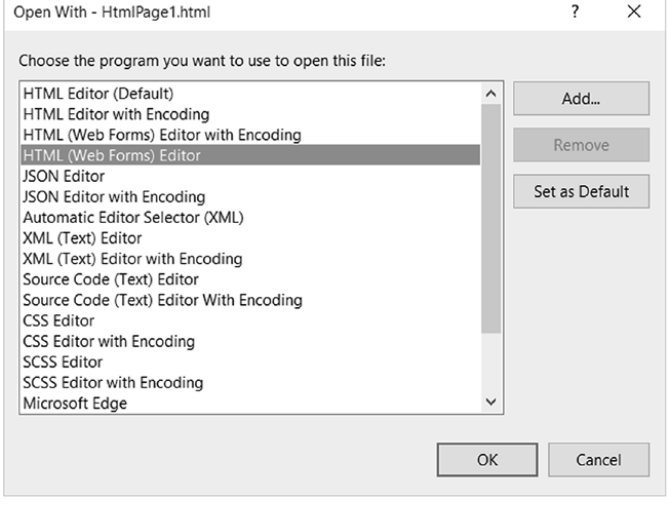



**Рис. B.5.** Выбор редактора HTML из Web Forms

Если щелкнуть на кнопке Set as Default (Установить по умолчанию), то больше не придется создавать решение, как описано здесь, потому что всегда будет применяться редактор HTML из Web Forms.

Щелчок на кнопке Split (Разделить) внизу окна редактора HTML приводит к тому, что нижняя панель будет отображать визуальную компоновку HTML, а верхняя — связанную разметку. Еще одно преимущество этого редактора заключается в том, что при выборе разметки или элемента HTML пользовательского интерфейса соответствующее представление подсвечивается. На рис. В.6 показан пример режима разделения в действии.

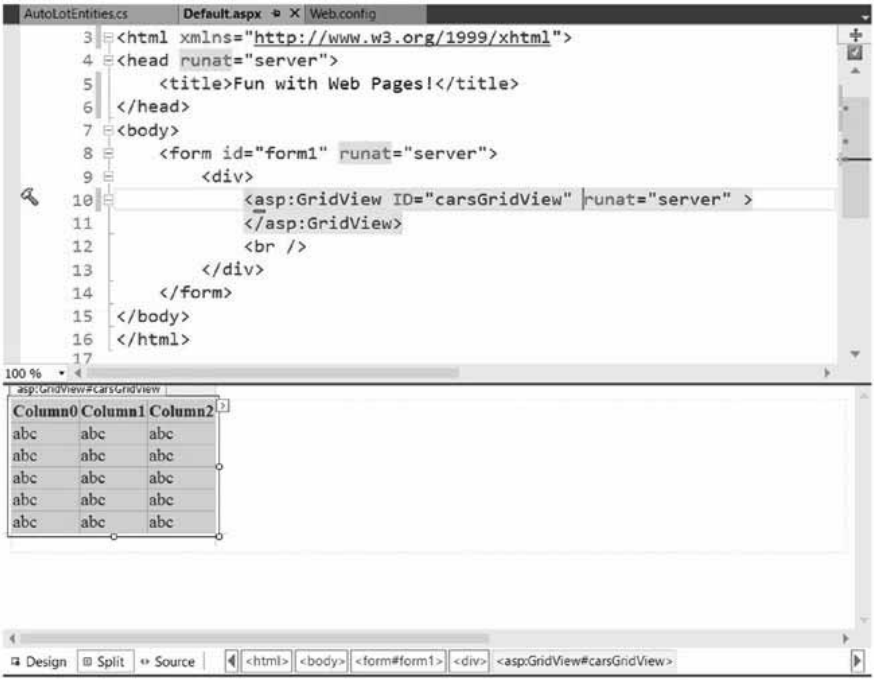



**Рис. В.6.** Редактор HTML из Web Forms в Visual Studio

Среда Visual Studio также позволяет редактировать общий внешний вид и поведение страницы `*.html` или отдельного элемента управления в `<form>` с использованием окна Properties (Свойства). Например, выбрав в раскрывающемся списке окна Properties элемент `DOCUMENT`, можно конфигурировать разнообразные аспекты страницы HTML (рис. В.7).

По мере использования окна Properties для конфигурирования какого-то аспекта веб-страницы IDE-среда будет соответствующим образом изменять разметку HTML. При проработке примеров в оставшихся приложениях для упрощения редактирования страниц HTML также можно применять IDE-среду.

## Построение формы HTML

Модифицируем раздел `<body>` первоначального файла, чтобы отображать литеральный текст, который приглашает пользователя ввести сообщение. Имейте в виду, что литеральное текстовое содержимое можно вводить и форматировать, набирая прямо в визуальном конструкторе разметки HTML.

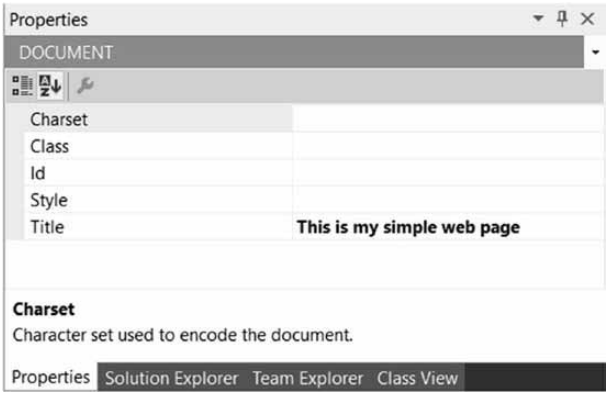



**Рис. B.7.** Окно Properties в Visual Studio может применяться для конфигурирования разметки HTML

Здесь дескриптор <h1> используется для установки веса заголовка, <p> — для блока абзаца и <i> — для курсивного текста:

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>This is my simple web page</title>
</head>
<body>
  <!-- Пригласить пользователя ввести текст -->
  <h1>Simple HTML Page</h1>
  <p>
    <br/>
    <i>Please enter a message</i>.
  </p>
  <form id="defaultPage">
  </form>
</body>
</html>
```

Теперь давайте создадим область ввода формы. В общем случае каждый элемент управления HTML описывается с применением атрибута id (для идентификации элемента в коде) и атрибута type (для указания того, какой элемент управления вводом должен быть помещен в объявление <form>).

Создаваемый пользовательский интерфейс будет содержать одно текстовое поле и две кнопки. Первая кнопка будет использоваться для запуска сценария клиентской стороны, а вторая — для сброса полей ввода формы в стандартные значения. Изменим форму HTML следующим образом:

```
<!-- Построить форму для получения информации о пользователе -->
<form id="defaultPage">
  <p>
    Your Message:
    <input id="txtUserMessage" type="text"/></p>
  <p>
    <input id="btnShow" type="button" value="Show!"/>
    <input id="btnReset" type="reset" value="Reset"/>
  </p>
</form>
```

Обратите внимание, что каждому элементу управления в атрибуте `id` назначается подходящий идентификатор (`txtUserMessage`, `btnShow` и `btnReset`). Кроме того, каждый элемент ввода имеет дополнительный атрибут `type`, который указывает, что элемент автоматически сбрасывает все поля в их начальные значения (`type="reset"`), принимает текстовый ввод (`type="text"`) или функционирует как простая кнопка клиентской стороны, ничего не отправляющая веб-серверу (`type="button"`).

Сохраним файл, щелкнем правой кнопкой мыши на поверхности визуального конструктора и выберем в контекстном меню пункт View in Browser (Просмотреть в браузере). На рис. В.8 показана текущая страница в новом браузере Microsoft Edge.

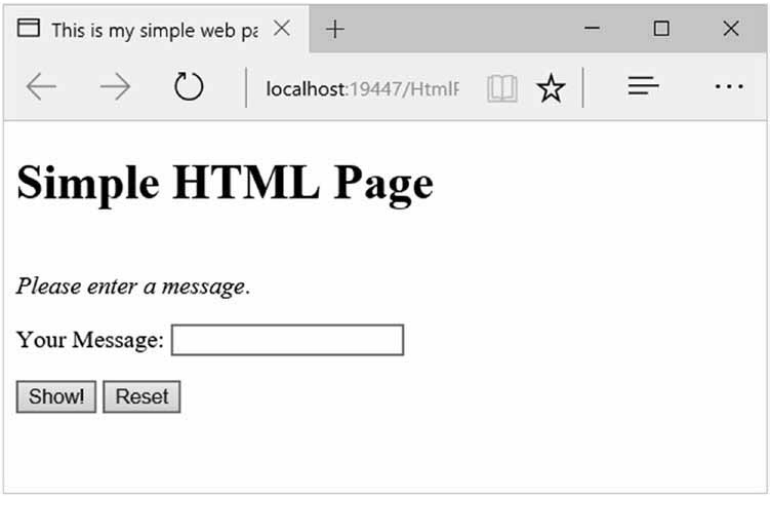



**Рис. В.8.** Простая страница HTML

**На заметку!** В случае выбора пункта View in Browser для файла HTML среда Visual Studio автоматически запустит веб-сервер IIS Express для размещения содержимого.

# Роль сценариев клиентской стороны

В дополнение к элементам графического пользовательского интерфейса файл `*.html` может содержать блоки сценарного кода, которые будут обрабатываться запрашивающим браузером. Существуют две главные причины использования сценариев на стороне клиента:

- проверка достоверности пользовательского ввода перед обратной отправкой веб-серверу;
- взаимодействие с объектной моделью документа (Document Object Model — DOM) браузера.

Относительно первой причины следует понимать, что врожденным изъяном веб-приложения является потребность выполнять частые полные обмены (называемые *обратным отправками*) с машиной сервера для обновления разметки HTML, визуализируемой в браузере. Хотя обратные отправки неизбежны, вы всегда должны стремиться к минимизации объема данных, передаваемых по сети. Один из приемов, которые экономят количество обратных отправок, связан с применением сценариев клиентской

стороны для проверки достоверности пользовательского ввода перед отправкой данных формы веб-серверу. В случае выявления ошибки, такой как отсутствие данных в обязательном поле, пользователя об этом можно уведомить без накладных расходов на обратную отправку веб-серверу. (В конце концов, нет ничего более раздражающего пользователей, чем отправка данных по медленному подключению лишь для того, чтобы получить инструкции по исправлению ошибок ввода!)

---

**На заметку!** Имейте в виду, что даже при выполнении проверки достоверности на стороне клиента (в целях сокращения времени реакции) проверка достоверности *также* должна происходить на самом веб-сервере. Это позволит гарантировать, что данные не были искажены после того, как клиент отправил их по сети. Элементы управления проверкой достоверности ASP.NET автоматически выполняют проверку достоверности на клиентской и серверной стороне (более подробно об них пойдет речь в приложении Г).

---

Сценарии клиентской стороны могут также использоваться для взаимодействия с лежащей в основе объектной моделью (DOM) самого браузера. Большинство коммерческих браузеров открывают доступ к набору объектов, которые можно задействовать для управления поведением браузера.

Когда браузер производит синтаксический анализ страницы HTML, он строит в памяти дерево объектов, представляющее все содержимое веб-страницы (формы, элементы управления вводом и т.д.). Браузеры предоставляют API-интерфейс под названием DOM, который открывает доступ к дереву объектов и позволяет программно модифицировать его содержимое. Например, можно написать код JavaScript, который выполняется в браузере для получения значений из определенных элементов управления, изменения цвета элемента, динамического добавления новых элементов управления к странице и т.п.

К большому сожалению, разные браузеры обычно предлагают сходные, *но не идентичные* объектные модели. Таким образом, блок сценарного кода клиентской стороны, который взаимодействует с DOM, в разных браузерах может работать неодинаково (и потому обязательно требует тестирования).

В ASP.NET имеется свойство `HttpRequest.Browser`, которое во время выполнения позволяет определять функциональные возможности браузера и устройства, откуда поступил текущий запрос. Данная информация позволяет максимально оптимизировать формируемый ответ HTTP. Но потребность в ней возникает редко, если только вы не реализуете специальные элементы управления, поскольку все стандартные веб-элементы управления в ASP.NET умеют соответствующим образом визуализировать себя в зависимости от типа браузера. Эта ценная способность известна как *адаптивная визуализация* и предоставляется в готовом виде для всех стандартных элементов управления ASP.NET.

Для написания сценариев клиентской стороны доступны разнообразные языки программирования, среди которых наиболее популярным является JavaScript. Важно помнить, что JavaScript ни в каких аспектах нельзя считать языком Java. Наряду с тем, что JavaScript и Java имеют кое в чем похожий синтаксис, язык JavaScript по сравнению с Java менее мощный. Но самое главное: все современные веб-браузеры поддерживают язык JavaScript, делая его естественным кандидатом для написания логики сценариев клиентской стороны.

## Пример сценария клиентской стороны

Чтобы проиллюстрировать роль сценариев клиентской стороны, давайте сначала выясним, как перехватывать события, отправляемые виджетами графического пользовательского интерфейса клиентской стороны. Для перехвата события щелчка на кнопке

Show! (Показать) добавим в определение элемента управления `btnShow` атрибут `onclick` и укажем в нем метод JavaScript по имени `btnShow_onclick()`:

```
<input id="btnShow" type="button" value="Show!"
       onclick="return btnShow_onclick()" />
```

Теперь поместите сразу после открывающего элемента `<head>` код функции JavaScript, которая вызывается, когда пользователь щелкает на кнопке. С помощью метода `alert()` отобразим на стороне клиента окно сообщения со значением из текстового поля, которое доступно через свойство `value`:

```
<script type="text/javascript">
// <![CDATA[
  function btnShow_onclick() {
    alert(window.txtUserMessage.value);
  }
// ]]>
</script>
```

Обратите внимание, что блок сценария помещен внутрь раздела `CDATA`. Причина проста: если страница попадет в браузер, который не поддерживает JavaScript, тогда код сценария будет воспринят как блок комментария и проигнорирован. Разумеется, страница может стать менее функциональной, но зато не вызовет ошибку во время визуализации в браузере. Теперь при просмотре страницы в браузере должна быть возможность вводить сообщение и видеть его в окне сообщения клиентской стороны (рис. B.9).

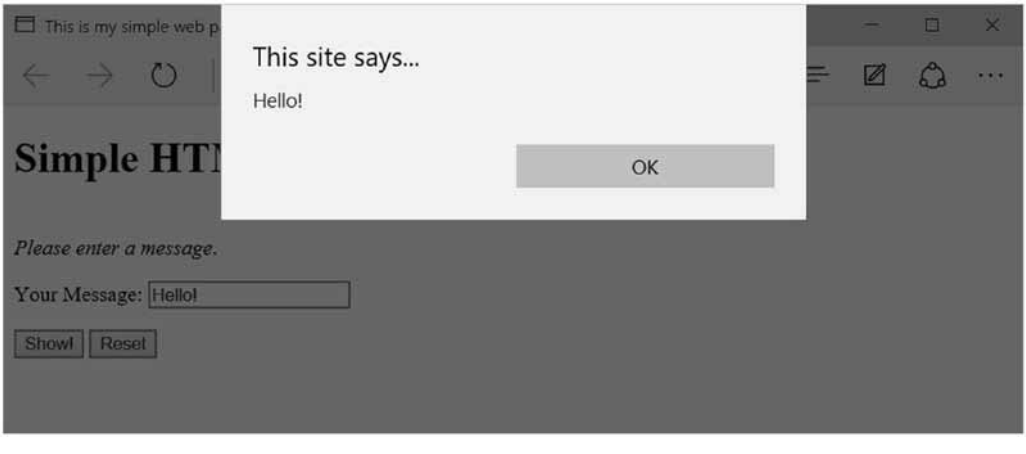



**Рис. B.9.** Вызов функции JavaScript клиентской стороны

Щелчок на кнопке Reset (Сброс) приводит к очистке текстового поля, потому что данная кнопка была определена с атрибутом `type="reset"`.

# Обратная отправка веб-серверу

Созданная простая страница HTML выполняет всю свою функциональность внутри браузера. Тем не менее, *реальная* веб-страница нуждается в обратной отправке по адресу ресурса на веб-сервере, одновременно передавая все введенные данные. После того, как ресурс серверной стороны получит отправленные данные, он может применять их для динамического построения подходящего HTTP-ответа.

Атрибут action в открывающем дескрипторе <form> указывает получателя входных данных формы. В число возможных получателей входят почтовые серверы, другие файлы HTML на веб-сервере, веб-службы REST, страницы Web Forms и т.д.

Помимо атрибута action вероятно также будет определена кнопка отправки (submit), щелчок на которой приведет к передаче данных формы веб-приложению через HTTP-запрос. В текущем примере в ней нет необходимости; однако ниже приведена разметка, где в открывающем дескрипторе <form> указан следующий атрибут:

```
<form id="defaultPage"
      action="http://localhost/Cars/MyAspNetPage.aspx" method="GET">
  <input id="btnPostBack" type="submit" value="Post to Server!"/>
  ...
</form>
```

Когда пользователь щелкает на кнопке отправки, данные формы отправляются файлу MyAspNetPage.aspx, расположенному по указанному URL. Если в качестве режима передачи задано method="GET", то данные формы присоединяются к строке запроса в виде набора пар "имя-значение", разделенных амперсандами. Вы наверняка уже видели данные подобного рода в браузере, которые выглядят так:

```
http://www.google.com/search?hl=en&source=hp&q=vikings&cts=1264370773666
&aq=f&aql=&aqi=g1g-z1g1g-z1g1g-z1g4&oq=
```

Другой метод передачи данных формы веб-серверу указывается как method="POST":

```
<form id="defaultPage"
     action="http://localhost/Cars/MyAspNetPage.aspx" method="POST">
  ...
</form>
```

В таком случае данные формы не присоединяются к строке запроса. При использовании метода POST данные формы не будут напрямую видны извне. Что более важно, данные POST не имеют ограничения по количеству символов, в то время как многие браузеры ограничивают длину запросов GET.

## Обратные отправки в Web Forms

При создании веб-сайтов на основе Web Forms инфраструктура самостоятельно позаботится о механизме отправки данных. Одним из многих преимуществ построения веб-сайтов с применением ASP.NET Web Forms является то, что модель программирования находится поверх стандартного цикла "запрос/ответ" HTTP, который относится к системе, управляемой событиями. Таким образом, вместо ручной установки атрибута action и определения кнопки отправки HTML можно просто обрабатывать события элементов управления Web Forms, используя стандартный синтаксис C#.

Применяя такую управляемую событиями модель, можно очень легко выполнять обратную отправку веб-серверу с помощью множества элементов управления. При необходимости это можно делать, когда пользователь щелкает на переключателе, на элементе в окне со списком, на каком-нибудь дне в элементе управления типа календаря и т.д. В каждом случае вы просто обрабатываете соответствующее событие, а исполняющая среда ASP.NET автоматически выдаст корректные данные HTML для отправки.

---

**Исходный код.** Веб-сайт SimpleWebPage доступен в подкаталоге Appendix_C.

# Обзор API-интерфейса Web Forms

К настоящему моменту краткий обзор разработки классических веб-приложений завершен, так что вы готовы углубиться в исследования Web Forms. Как и можно было ожидать, в каждой новой версии платформы .NET к API-интерфейсам программирования веб-приложений добавлялась дополнительная функциональность, и утверждение справедливо также в отношении .NET 4.7. Независимо от версии .NET, с которой вы работаете, перечисленные ниже возможности доступны всем веб-приложениям, основанным на ASP.NET Web Forms.

- Инфраструктура ASP.NET предоставляет модель отделенного кода, которая позволяет отделять логику презентации (разметка HTML) от бизнес-логики (код C#).
- Страницы ASP.NET кодируются с использованием языков программирования .NET, а не языков сценариев серверной стороны. Файлы кода компилируются в допустимые .NET-сборки `*.dll` (которые транслируются в намного более быстродействующий исполняемый код).
- Элементы управления Web Forms могут применяться для построения пользовательского веб-интерфейса в манере, похожей на построение настольных приложений Windows.
- Приложения Web Forms могут использовать любые сборки из библиотек базовых классов .NET и конструируются с применением объектно-ориентированных технологий, рассматриваемых в данной книге (классы, интерфейсы, структуры, перечисления и делегаты).
- Приложения Web Forms можно легко конфигурировать с помощью конфигурационного файла веб-приложения (`Web.config`).

Прежде всего, здесь необходимо отметить, что пользовательский интерфейс веб-страницы Web Forms может быть сконструирован с использованием разнообразных *веб-элементов управления*. В отличие от типичного элемента управления HTML веб-элементы управления функционируют на веб-сервере и выдают в HTTP-ответе корректные дескрипторы HTML. Одно лишь это является огромным преимуществом Web Forms, потому что значительно сокращает объем разметки HTML, которая должна быть написана вручную. В качестве небольшого примера предположим, что внутри веб-страницы Web Forms определен следующий элемент управления Web Forms:

```
<asp:Button ID="btnMyButton" runat="server" Text="Button" BorderColor="Blue"
            BorderStyle="Solid" BorderWidth="5px" />
```

Вы узнаете детали объявления элементов управления Web Forms чуть позже, а пока обратите внимание, что многие атрибуты элемента управления `<asp:Button>` выглядят очень похожими на свойства, которые встречались в примерах WPF. То же самое верно для всех элементов управления Web Forms, поскольку при создании инструментального набора веб-элементов управления в Microsoft виджеты намеренно проектировались так, чтобы иметь внешний вид и поведение, подобные своим настольным аналогам.

Если теперь браузер обратится к файлу `.aspx`, содержащему указанный элемент управления, то данный элемент выдаст в выходной поток следующее объявление HTML:

```
<input type="submit" name="btnMyButton" value="Button" id="btnMyButton"
       style="border-color:Blue;border-width:5px;border-style:Solid;" />
```

Обратите внимание, что веб-элемент управления выпускает стандартную разметку HTML, которая может быть визуализирована в любом браузере. С учетом этого важно понимать, что применение элементов управления Web Forms никак не привязывает к

семейству операционных систем Microsoft или к браузеру Microsoft Internet Explorer/Edge. Страницу Web Forms можно просматривать в среде любой операционной системы и в любом браузере (включая портативные устройства, такие как Apple iPhone, Android и Windows Phone).

В предшествующем списке возможностей было указано, что приложение Web Forms будет компилироваться в сборку .NET. Таким образом, веб-проекты ничем не отличаются от любой .NET-сборки `.dll`, которая строилась в примерах, рассмотренных в книге. Скомпилированное веб-приложение будет состоять из кода CIL, манифеста сборки и метаданных типов. Это дает несколько крупных преимуществ, наиболее заметными из которых являются выигрыш в производительности, строгая типизация и возможность микроуправления со стороны CLR (например, сборка мусора и т.д.).

Наконец, приложения Web Forms предоставляют программную модель, посредством которой разметку страницы можно отделять от связанной кодовой базы C# с применением *файлов кода*. В случае использования файлов кода введенная разметка будет отображаться на полноценную объектную модель, которая объединяется с файлом кода C# через объявления частичных классов.

## Основные возможности Web Forms 2.0 и последующих версий

Версия ASP.NET 1.0 была крупным шагом в правильном направлении, а ASP.NET 2.0 предоставила множество дополнительных средств, которые помогли перейти от построения динамических *веб-страниц* к построению многофункциональных *веб-сайтов*. Ниже приведен частичный список основных средств.

- Появление веб-сервера разработки ASP.NET (означающего, что разработчики больше не нуждаются в наличии полной версии IIS, установленной на машинах разработки). Теперь он заменен IIS Express.
- Большое количество новых веб-элементов управления, которые обеспечили поддержку во многих сложных ситуациях (элементы управления навигацией, элементы управления безопасностью, новые элементы управления привязкой данных и т.д.).
- Появление мастер-страниц, которые позволяют разработчикам присоединять общие области пользовательского интерфейса к набору связанных страниц.
- Поддержка тем, которые предлагают декларативный способ изменения внешнего вида и поведения всего веб-приложения на веб-сервере.
- Поддержка веб-частей (Web Parts), которые позволяют конечным пользователям настраивать внешний вид и поведение веб-страницы и сохранять настройки для последующего применения (подобно порталам).
- Появление веб-ориентированной утилиты конфигурации и управления, которая обслуживает разнообразные файлы `Web.config`.

Помимо веб-сервера разработки ASP.NET одним из самых крупных нововведений ASP.NET 2.0 было появление *мастер-страниц*. Как известно, большинство веб-сайтов поддерживают согласованный внешний вид и поведение для всех страниц сайта. Посмотрите на коммерческий веб-сайт вроде `www.amazon.com`. Каждая страница содержит одни и те же элементы, такие как общий заголовок, общий нижний колонтитул, общее меню навигации и т.п.

С использованием мастер-страниц можно моделировать общую функциональность и определять *места заполнения* для подключения других файлов `.aspx`. Это существенно облегчает изменение общего вида сайта (перемещение панели навигации в другое место, смену логотипа в заголовке и т.д.) за счет простой модификации мастер-страницы, оставляя другие файлы `.aspx` незатронутыми.

**На заметку!** Мастер-страницы настолько удобны, что в Visual Studio 2010 и последующих версиях все новые веб-проекты Web Forms по умолчанию включают мастер-страницу.

В ASP.NET 2.0 также было добавлено много новых веб-элементов управления, в том числе элементы управления, которые автоматически поддерживают общие средства безопасности (вход на сайт, восстановление пароля и т.д.), элементы управления, позволяющие укладывать навигационную структуру поверх набора связанных файлов `.aspx`, и другие элементы управления, выполняющие сложные операции привязки данных, где необходимые SQL-запросы могут генерироваться с применением набора элементов управления Web Forms.

## Основные возможности Web Forms 3.5 (и .NET 3.5 SP1) и последующих версий

Следует отметить, что в версии .NET 3.5 приложения Web Forms получили возможность задействовать модель программирования LINQ (также появившуюся в .NET 3.5) и перечисленные далее веб-ориентированные средства.

- Поддержка привязки данных для классов ADO.NET Entity Framework (см. главу 22).
- Поддержка ASP.NET Dynamic Data — инфраструктуры, появившейся благодаря Ruby on Rails, которая может использоваться для построения веб-приложений, управляемых данными. Она открывает доступ к таблицам базы данных за счет их кодирования в URI веб-службы ASP.NET, а для данных в таблицах автоматически генерируется разметка HTML.
- Интегрированная поддержка разработки в стиле Ajax, которая по существу позволяет выполнять обратные микро-отправки для как можно более быстрого обновления части веб-страницы.

Шаблоны проектов ASP.NET Dynamic Data, появившиеся в .NET 3.5 Service Pack 1, предлагают новую модель для построения сайтов, которые в высокой степени управляются реляционной базой данных. Конечно, большинство веб-сайтов в какой-то мере будут нуждаться во взаимодействии с базами данных, но проекты ASP.NET Dynamic Data тесно связаны с ADO.NET Entity Framework и непосредственно ориентированы на быструю разработку сайтов, управляемых данными (аналогичных сайтам, которые строятся с помощью Ruby).

## Основные возможности Web Forms 4.0

В версии .NET 4.0 к платформе веб-разработки Microsoft были добавлены дополнительные возможности. Вот список некоторых наиболее заметных веб-ориентированных средств.

- Возможность сжатия данных “состояния представления” с применением стандарта GZIP.
- Библиотека JQuery включена для инфраструктур Web Forms и MVC.
- Обновленные определения браузеров для обеспечения корректной визуализации страниц ASP.NET в новых браузерах и устройствах (Google Chrome, Apple iPhone, Windows Phone, Android и т.д.).
- Возможность настройки вывода из элементов управления проверкой достоверности с помощью каскадных таблиц стилей (CSS).

- Включение в библиотеку ASP.NET элемента управления `Chart`, который позволяет создавать страницы ASP.NET с наглядными диаграммами для сложного статистического или финансового анализа.
- Поддержка шаблонов проектов ASP.NET Model View Controller, которая уменьшает зависимость между уровнями приложения за счет использования шаблона MVC (Model-View-Controller — модель-представление-контроллер). Это совершенно другой подход к разработке веб-сайтов, который немного напоминает рассматриваемую здесь модель программирования Web Forms.

Несмотря на то что список впечатляет (и он включает лишь подмножество новых средств), работа, которую проделали специалисты из Microsoft в отношении Web Forms для ASP.NET 4.5 заставила многих разработчиков возвратиться к Web Forms из ASP.NET MVC (см. главу 29).

## Основные возможности Web Forms 4.5 и Web Forms 4.6

Двумя главными областями, которым уделялось внимание в версии .NET 4.5, были улучшения производительности и перенос многих возможностей ASP.NET MVC в инфраструктуру Web Forms. Ниже представлен частичный список нововведений в Web Forms 4.5 и Web Forms 4.6.

### *Возможности, добавленные в Web Forms 4.5*

- Многочисленные обновления для поддержки HTML 5.0.
- Интеграция с новыми асинхронными языковыми возможностями C# и VB.
- Объявление типа данных, к которому планируется привязывать элемент управления, с применением нового свойства `ItemType`, что делает возможными строго типизированные элементы управления, поддержку IntelliSense и многое другое.
- Привязка модели, которая означает возможность отображения данных из страницы прямо на параметры типа метода.
- Проверка достоверности клиентской стороны теперь интегрирована с JQuery, позволяя писать более ясный код проверки.
- Стали доступны дополнительные возможности проверки достоверности посредством аннотаций данных, которые являются атрибутами в классах моделей.
- Дополнительная защита от атак межсайтовыми сценариями с включением (по умолчанию) библиотеки AntiXSS.
- Сокращение размеров файлов (JavaScript и CSS) с помощью минификации (уменьшающей размеры файлов за счет сжатия текста внутри файлов).
- Сокращенное количество обращений браузера за счет объединения файлов в один посредством упаковки.
- Возможность откладывания проверки достоверности запросов, что позволяет отправлять потенциально небезопасное содержимое (следует использовать с осторожностью).
- Возможность применения более одного серверного кода при компиляции приложений Web Forms.

### *Возможности, добавленные в Web Forms 4.6*

- Поддержка нового высокоскоростного протокола HTTP2 (в настоящий момент он доступен только для защищенных приложений на IIS).

- Включение новых средств C# 6 с использованием компилятора Roslyn.
- Возможность применения `async/await` в функциях привязки моделей.

Вы наверняка согласитесь с тем, что набор средств Web Forms довольно широк (к тому же, этот API-интерфейс содержит намного больше возможностей, чем было здесь перечислено). По правде говоря, если попытаться раскрыть все средства Web Forms, то объем книги легко увеличился бы вдвое (а то и втрое). Поскольку это нереально, мы рассмотрим только базовые средства ASP.NET, которые вероятно будут использоваться вами ежедневно. Описание средств, которые в книге не раскрыты, находится в документации .NET Framework 4.7 SDK.

---

**На заметку!** Если нужно исчерпывающее руководство по разработке веб-приложений в ASP.NET, тогда рекомендуется почитать книгу *ASP.NET 4.5 с примерами на C# 5.0 для профессионалов, 5-е изд.* (ИД "Вильямс").

---

# Построение однофайловой веб-страницы Web Forms

Страница Web Forms может быть сконструирована с применением двух основных подходов, первый из которых предусматривает построение единственного файла `.aspx`, содержащего смесь кода серверной стороны и разметки HTML. В случае подхода с *моделью однофайловой страницы* код серверной стороны помещается внутрь области `<script>`, но сам код *не* является сценарием (например, на VBScript или JavaScript). Взамен код в блоке `<script>` пишется на выбранном языке .NET (C#, Visual Basic и т.д.).

При построении веб-страницы, которая содержит очень мало кода (но значительный объем статической разметки HTML), модель однофайловой страницы может оказаться проще в работе, потому что она позволяет видеть код и разметку в едином файле `.aspx`. Кроме того, помещение процедурного кода и разметки HTML в единственный файл `.aspx` обеспечивает ряд других преимуществ.

- Страницы, написанные с использованием однофайловой модели, несколько легче развертывать или отправлять другим разработчикам.
- Поскольку нет никаких зависимостей между многочисленными файлами, однофайловую страницу проще переименовывать.
- Немного облегчается манипулирование файлами в системе управления исходным кодом, т.к. все действия производятся с единственным файлом.

Недостаток модели однофайловой страницы связан с тем, что она приводит к появлению излишне сложных файлов, поскольку разметка пользовательского интерфейса и программная логика находятся в одном месте. Тем не менее, мы начнем экскурс в Web Forms с исследования модели однофайловой страницы.

Наша цель заключается в построении файла `.aspx`, который отображает содержимое таблицы `Inventory` базы данных `AutoLot` (созданной в главе 21) с применением Entity Framework. Запустим Visual Studio, выберем пункт меню File⇨New Project (Файл⇨Создать проект), после чего откроется диалоговое окно New Project (Новый проект). В узле Visual C# древовидного представления слева выберем элемент Web, в центральной панели укажем в качестве типа проекта ASP.NET Web Application (Веб-приложение ASP.NET) и введем `SinglePageModel` в поле Name (Имя), как показано на рис. B.10.

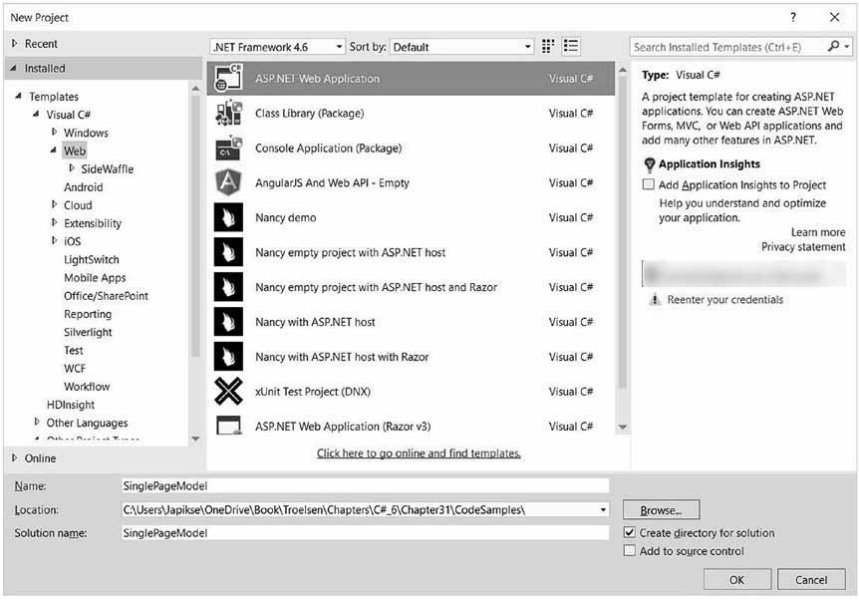



**Рис. В.10.** Выбор типа проекта ASP.NET Web Application в диалоговом окне New Project

После щелчка на кнопке OK отобразится обновленное диалоговое окно New ASP.NET Project (Новый проект ASP.NET). В области ASP.NET 4.6 Templates (Шаблоны ASP.NET 4.6) выберем шаблон Empty (Пустой). Оставим флажки Web Forms, MVC и Web API в области Add folders and core references for (Добавить папки и основные ссылки для), а также флажок Add unit tests (Добавить модульные тесты) неотмеченными. Снимем отметку с флажка Host in the cloud (Разместить в облаке) в области Microsoft Azure, если он отмечен (рис. В.11).

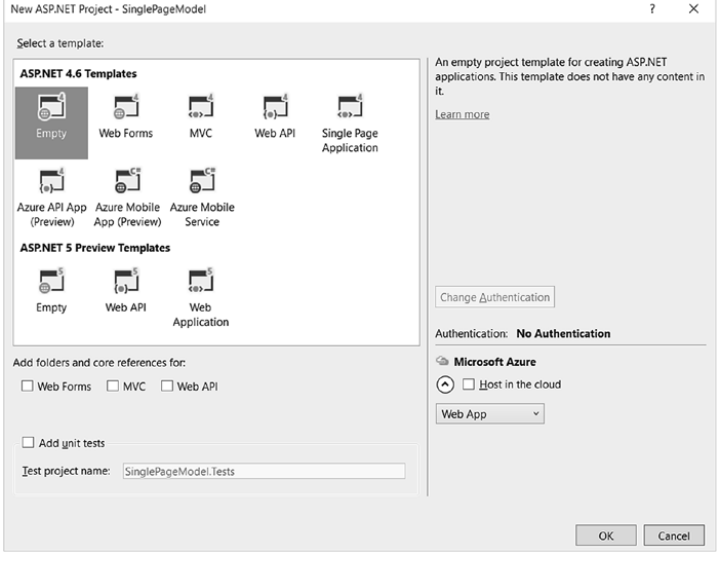



**Рис. В.11.** Выбор шаблона проекта Empty

Добавим в проект новую веб-форму, выбрав пункт меню Project⇨Add New Item (Проект⇨Добавить новый элемент). В древовидном представлении слева должен быть выбран узел Web, а в нем Web Forms. Назначим файлу имя `Default.aspx`.

## Добавление ссылки на сборку `AutoLotDAL.dll`

С помощью проводника Windows скопируем папку `AutoLotDAL` из подкаталога `Chapter_22` (либо из папки `SinglePageModel` в подкаталоге `Appendix_C`). Добавим проект к решению, щелкнув правой кнопкой мыши на имени решения, выбрав в контекстном меню пункт Add⇨Existing Project (Добавить⇨Существующий проект) и указав проект `AutoLotDAL`. Затем добавим ссылку на проект `AutoLotDAL`, для чего щелкнем правой кнопкой мыши на узле References (Ссылки) в проекте `SinglePageModel` и выберем `AutoLotDAL`.

В проект `AutoLotDAL` понадобится внести небольшое изменение, связанное с пространством имен `System.Web`. При вызове конструктора класса `DatabaseLogger` не указан каталог, что приведет к отказу в работе кода на веб-сайте из-за нехватки разрешений. Ситуацию необходимо исправить, указав физический каталог веб-сайта. В пространстве имен `System.Web` имеется серверная переменная по имени `HttPRuntime.AppDomainAppPath`, которая содержит физический каталог веб-сайта. Откроем файл `AutoLotEntities.cs` из папки `EF`, добавим оператор `using` для `System.Web` и изменим инициализатор `DatabaseLogger` следующим образом:

```
static readonly DatabaseLogger DatabaseLogger =
  new DatabaseLogger($"{HttpRuntime.AppDomainAppPath}/sqllog.txt");
```

Такое изменение обеспечивает создание журнального файла в том же каталоге, где находится веб-сайт, устраняя проблему с разрешениями.

Добавим к веб-проекту инфраструктуру Entity Framework, щелкнув правой кнопкой мыши на имени решения в окне Solution Explorer, выбрав в контекстном меню пункт Manage NuGet Packages (Управление пакетами NuGet) и указав Entity Framework. Понадобится также модифицировать файл `Web.config` (аналогичен файлам `App.config`, с которыми приходилось работать в книге), как показано ниже (в строке подключения должно быть указано соответствующее имя экземпляра SQL Server):

```
<configuration>
  <configSections>
    <!-- Дополнительные сведения о конфигурации Entity Framework
      доступны по адресу http://go.microsoft.com/fwlink/?LinkID=237468. -->
    <section name="entityFramework"
      type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c5619
34e089"
      requirePermission="false"/>
  </configSections>
  <system.web>
    <compilation debug="true" targetFramework="4.6"/>
    <httpRuntime targetFramework="4.6"/>
  </system.web>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.
LocalDbConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="mssqllocaldb"/>
      </parameters>
    </defaultConnectionFactory>
```

```
      <providers>
        <provider invariantName="System.Data.SqlClient" type="System.Data.
Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer"/>
      </providers>
    </entityFramework>
    <connectionStrings>
      <add name="AutoLotConnection"
        connectionString="data source=.\SQLEXPRESS2014;initial
catalog=AutoLot;integrated security=True;MultipleActiveResultSets=True;
App=EntityFramework"
        providerName="System.Data.SqlClient"/>
    </connectionStrings>
  </configuration>
```

## Проектирование пользовательского интерфейса

Откроем файл `Default.aspx`, щелкнем на вкладке Design (Схема), перейдем на вкладку Data (Данные) панели инструментов Visual Studio и перетащим на поверхность визуального конструктора страницы элемент управления `GridView`, поместив его между открывающим и закрывающим элементами `form`. Визуальный конструктор заполняет `GridView` произвольными данными, чтобы дать представление о том, как будет выглядеть страница. С помощью окна Properties установим разнообразные визуальные свойства. Отыщем в разметке страницы раздел `<form>`. Обратите внимание, что веб-элемент управления определяется с использованием дескриптора `<asp:>`. После префикса `asp` указано имя элемента управления Web Forms (`GridView`). Перед закрывающим дескриптором элемента находится последовательность пар "имя-значение", которые соответствуют настройкам, доступным в окне Properties:

```
<form id="form1" runat="server">
<div>
  <asp:GridView ID="carsGridView" runat="server">
  </asp:GridView>
</div>
</form>
```

Элементы управления Web Forms (а также атрибут `runat="server"`) будут подробно рассматриваться в приложении Г. Пока просто запомните, что веб-элементы управления — это объекты, обрабатываемые на веб-сервере, который автоматически выпускает их представление HTML в исходящем HTTP-ответе. Помимо такого важного преимущества элементы управления Web Forms имитируют модель программирования настольных приложений в том, что имена свойств, методов и событий обычно воспроизводят их аналоги из Windows Forms/WPF.

## Добавление логики доступа к данным

Теперь добавим к дескриптору `asp:GridView` атрибут `ItemType` со значением `AutoLotDAL.Models.Inventory`. Такая новая возможность, появившаяся в .NET 4.5, обеспечивает поддержку строго типизированных списковых элементов управления в ASP.NET Web Forms и позволяет средству IntelliSense распознавать классы, доступные в решении.

Добавим атрибут `SelectMethod` со значением `GetData`. Он также был введен в версии .NET 4.5 и устанавливает метод, который будет выполняться, когда элемент управления визуализируется для получения данных, заполняющих списковый элемент управления. Ниже показана обновленная разметка:

```
<asp:GridView ID="carsGridView" runat="server"
  ItemType="AutoLotDAL.Models.Inventory"
  SelectMethod="GetData" >
</asp:GridView>
```

Создадим метод `GetData()` внутри дескриптора `<script>` страницы. В методе `GetData()` вызовем метод `InventoryRepo.GetAll()`. С помощью директив `<%@ Import ... %>` импортируем пространства имен `AutoLotDAL.Models` и `AutoLotDAL.Repos`. Код должен выглядеть следующим образом:

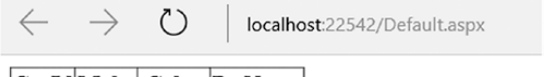
```
<!-- Находится в начале файла, после директивы Page -->
<%@ Import Namespace="AutoLotDAL.Models" %>
<%@ Import Namespace="AutoLotDAL.Repos" %>

<!-- Находится где-то в файле, перед определением элемента управления
GridView -->
<script runat="server">
  public IEnumerable<Inventory> GetData()
  {
    return new InventoryRepo().GetAll();
  }
</script>
```

**На заметку!** При построении страницы с применением однофайловой модели необходимо использовать только директиву `<%@ Import %>`. В случае стандартного подхода с файлом кода пространства имен включаются в этот файл с помощью ключевого слова `using` языка C#. То же самое касается директивы `<%@ Assembly %>`.

Прежде чем приступить к исследованию формата файла `*.aspx` давайте произведем тестовый запуск. Сохраним файл `.aspx`. Щелкнем на значке Run (Запустить) или нажмем <F5>, что приведет к запуску веб-сервера IIS Express, на котором разместится страница.

Во время обслуживания страницы выполняется метод, указанный в атрибуте `SelectMethod`, который загружает данные в элемент управления `GridView`. Результат приведен на рис. B.12.




| CarId | Make | Color | PetName |
|---|---|---|---|
| 1 | VW | Black | Zippy |
| 2 | Ford | Rust | Rusty |
| 3 | Saab | Black | Mel |
| 4 | Yugo | Yellow | Clunker |
| 5 | BMW | Black | Bimmer |
| 6 | BMW | Green | Hank |
| 7 | BMW | Pink | Pinky |
| 13 | Pinto | Black | Pete |

**Рис. B.12.** ASP.NET предоставляет декларативную модель привязки данных

Текущий пользовательский интерфейс довольно скромен. Чтобы улучшить его, выберем элемент `GridView` на поверхности визуального конструктора и в контекстном меню (открывающемся по щелчку на небольшой стрелке в правом верхнем углу элемента) выберем пункт Auto Format (Автоформат), как показано на рис. B.13.

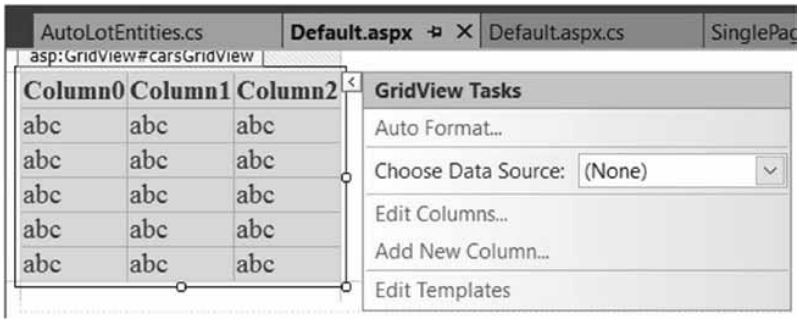



**Рис. В.13.** Конфигурирование элемента управления `GridView`

В открывшемся диалоговом окне выберем подходящий шаблон (скажем, Slate). После щелчка на кнопке OK просмотрим сгенерированное объявление элемента управления, которое стало более развитым, нежели ранее:

```
<asp:GridView ID="carsGridView" runat="server"
     ItemType="AutoLotDAL.Models.Inventory"
     SelectMethod="GetData" BackColor="White" BorderColor="#E7E7FF"
     BorderStyle="None" BorderWidth="1px" CellPadding="3"
     GridLines="Horizontal" >
  <AlternatingRowStyle BackColor="#F7F7F7" />
  <FooterStyle BackColor="#B5C7DE" ForeColor="#4A3C8C" />
  <HeaderStyle BackColor="#4A3C8C" Font-Bold="True" ForeColor="#F7F7F7" />
  <PagerStyle BackColor="#E7E7FF" ForeColor="#4A3C8C" HorizontalAlign="Right"/>
  <RowStyle BackColor="#E7E7FF" ForeColor="#4A3C8C" />
  <SelectedRowStyle BackColor="#738A9C" Font-Bold="True" ForeColor="#F7F7F7" />
  <SortedAscendingCellStyle BackColor="#F4F4FD" />
  <SortedAscendingHeaderStyle BackColor="#5A4C9D" />
  <SortedDescendingCellStyle BackColor="#D8D8F0" />
  <SortedDescendingHeaderStyle BackColor="#3E3277" />
</asp:GridView>
```

Если снова запустить приложение и щелкнуть на кнопке, то отобразится более интересный пользовательский интерфейс (рис. В.14).

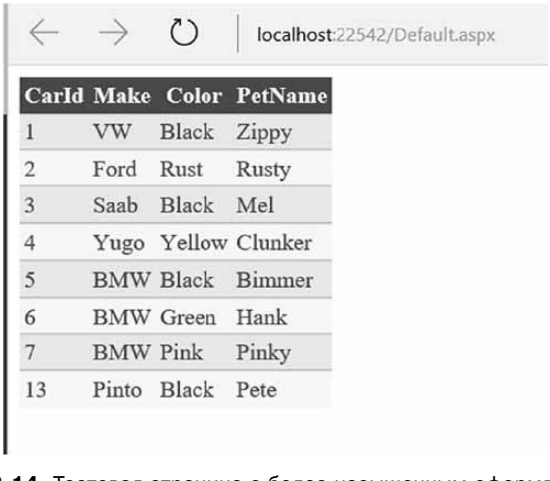



**Рис. В.14.** Тестовая страница с более насыщенным оформлением

Просто, не правда ли? Разумеется, как всегда, сложности кроются в деталях, поэтому давайте немного углубимся в состав файла .aspx, начав с выяснения роли директивы <%@ Page ... %>. Имейте в виду, что рассмотренные здесь темы будут применимы к рекомендуемой модели с *файлом кода*, которая описана далее в приложении.

## Роль директив ASP.NET

Файл .aspx обычно начинается с набора *директив*. Директивы ASP.NET всегда помечаются маркерами <%@ ... %> и могут содержать разнообразные атрибуты, которые информируют исполняющую среду ASP.NET о том, как следует обрабатывать конкретную директиву.

Каждый файл .aspx должен содержать минимум директиву <%@ Page %>, которая используется для определения управляемого языка, применяемого внутри страницы (посредством атрибута language). Кроме того, директива <%@ Page %> может определять имя связанного файла отделенного кода (рассматривается ниже) и т.д. Наиболее интересные атрибуты директивы <%@ Page %> кратко описаны в табл. В.1.

**Таблица В.1. Избранные атрибуты директивы <%@ Page %>**

| Атрибут | Описание |
|---|---|
| CodePage | Указывает имя связанного файла отделенного кода |
| EnableTheming | Указывает, поддерживают ли элементы управления на странице .aspx темы ASP.NET |
| EnableViewState | Указывает, поддерживается ли состояние представления между запросами страницы (более подробно это свойство рассматривается в приложении Д) |
| Inherits | Определяет класс в файле отделенного кода, от которого наследуется страница; может быть любым классом, производным от System.Web.UI.Page |
| MasterPageFile | Устанавливает мастер-страницу, используемую в сочетании с текущей страницей .aspx |
| Trace | Указывает, включена ли трассировка |

В дополнение к директиве <%@ Page %> файл .aspx может содержать различные директивы <%@ Import %> для явного указания пространств имен, требуемых текущей страницей, и директивы <%@ Assembly %> для описания внешних библиотек кода, которые используются сайтом (и обычно расположены в папке \bin веб-сайта).

В данном примере было заявлено применение типов из пространств имен Models и Repos сборки AutoLotDAL.dll. Если необходимо использовать дополнительные пространства имен .NET, тогда нужно просто указать несколько директив <%@ Import %>/ <%@ Assembly %>.

Помимо директив <%@ Page %>, <%@ Import %> и <%@ Assembly %> в ASP.NET определено несколько других директив, которые могут появляться в файле .aspx; они будут обсуждаться позже. Примеры применения других директив вы найдете в оставшихся приложениях.

## Анализ блока <script>

В однофайловой модели страницы файл .aspx может содержать логику сценариев серверной стороны, которая выполняется на веб-сервере. В таком случае *критически важно*, чтобы все блоки кода серверной стороны были определены для выполнения на сервере с использованием атрибута runat="server". Если атрибут runat="server" не

указан, то исполняющая среда считает, что был написан блок сценария *клиентской стороны*, который предназначен для встраивания в исходящий HTTP-ответ, и генерирует исключение. Ниже показан правильный блок `<script>` серверной стороны:

```
<script runat="server">
  public IEnumerable<Inventory> GetData()
  {
    return new InventoryRepo().GetAll();
  }
</script>
```

**На заметку!** Все элементы управления Web Forms должны иметь в открывающем дескрипторе атрибут `runat="server"`. В противном случае они не будут генерировать свою разметку HTML в исходящий HTTP-ответ.

## Анализ объявлений элементов управления ASP.NET

Последний интересный момент в первом примере связан с объявлением веб-элемента управления `GridView`. Подобно классическому коду ASP и низкоуровневой разметке HTML виджеты Web Forms помещаются внутрь элементов `<form>`. Однако открывающий дескриптор `<form>` помечен атрибутом `runat="server"`, а элементы управления снабжены дескрипторным префиксом `asp:`. Любой элемент с таким префиксом является членом библиотеки элементов управления ASP.NET и имеет соответствующее представление в виде класса C# внутри некоторого пространства имен .NET библиотеки базовых классов .NET. Вот как выглядит разметка:

```
<form id="form1" runat="server">
  <div>
    <asp:GridView ID="carsGridView" runat="server"
         ItemType="AutoLotDAL.Models.Inventory"
         SelectMethod="GetData" >
    </asp:GridView>
  </div>
</form>
```

Пространство имен `System.Web.UI.WebControls` сборки `System.Web.dll` содержит большинство элементов управления Web Forms. Открыв браузер объектов Visual Studio, можно обнаружить, скажем, элемент управления `DataGrid` (рис. B.15).

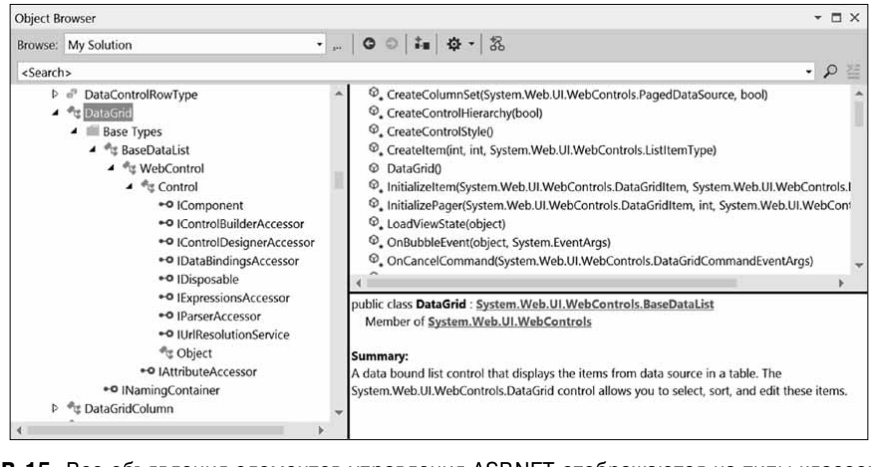



**Рис. B.15.** Все объявления элементов управления ASP.NET отображаются на типы классов .NET

Как видите, на самой вершине цепочки наследования для любого элемента управления Web Forms находится класс `System.Object`. Родительский класс `WebControl` — это общий базовый класс для всех элементов управления ASP.NET, который определяет все общие свойства пользовательского интерфейса (`BackColor`, `Height` и т.д.). Класс `Control` также очень распространен в инфраструктуре, но в нем определены члены, больше ориентированные на инфраструктуру (привязка данных, состояние представления и т.п.), а не на внешний вид и поведение дочерних элементов. Дополнительные сведения об этих классах вы узнаете в приложении Д.

---

**Исходный код.** Веб-сайт `SinglePageModel` доступен в подкаталоге `Appendix_C`.

---

# Построение веб-страницы ASP.NET с использованием файлов кода

Хотя однофайловая модель временами бывает полезной, стандартный подход, принятый в Visual Studio (при создании нового веб-проекта), предусматривает применение *отделенного кода*, что позволяет разносить программный код серверной стороны и логику презентации HTML по двум разным файлам. Такая модель работает довольно хорошо, когда страницы содержат существенный объем кода или когда созданием одного веб-сайта занимается множество разработчиков. Модель отделенного кода дает и другие преимущества.

- Поскольку страницы с отделенным кодом обеспечивают четкое разделение разметки HTML и кода, можно организовать параллельную работу дизайнеров над разметкой и программистов над кодом C#.
- Код не виден дизайнерам страниц и прочему персоналу, который имеет дело только с разметкой страницы (как и можно было догадаться, специалистам по разметке HTML не всегда интересны детали кода C#).
- Файлы кода можно использовать во множестве файлов `.aspx`.

Независимо от выбранного подхода *никакой* разницы в производительности не будет. На самом деле многие приложения Web Forms выигрывают, когда задействованы оба подхода. Для демонстрации модели с отделенным кодом давайте воссоздадим предыдущий пример с применением шаблона пустого веб-сайта в Visual Studio. Выберем пункт меню File⇨New Project (Файл⇨Создать проект), затем ASP.NET Web Application (Веб-приложение ASP.NET) и, наконец, шаблон Empty (Пустой) в группе ASP.NET 4.6 Templates (Шаблоны ASP.NET 4.6).

Теперь с использованием пункта меню Project⇨Add New Item (Проект⇨Добавить новый элемент) вставим новый элемент Web Form (Веб-форма) по имени `Default.aspx`. Создадим в визуальном конструкторе пользовательский интерфейс, состоящий из одного элемента управления `GridView`, и настроим его в окне Properties по собственному усмотрению. При желании можно просто скопировать в новый файл `.aspx` объявления элементов управления из примера `SinglePageModel`. Учитывая, что разметка здесь в точности та же самая, повторно она не приводится (понадобится только поместить объявления элементов управления между дескрипторами `<form>` и `</form>`).

Обратите внимание, что директива `<%@ Page %>`, применяемая в модели файла кода, имеет несколько атрибутов:

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="CodeBehindPageModel.Default" %>
```

Атрибут `CodeFile` используется для указания связанного внешнего файла, который содержит код логики для страницы. По умолчанию такие файлы отделенного кода именуются за счет добавления суффикса `.cs` к имени файла `.aspx` (в рассматриваемом примере получается `Default.aspx.cs`). Заглянув в окно Solution Explorer, легко заметить, что файл отделенного кода находится в подузле под значком веб-формы (рис. B.16).

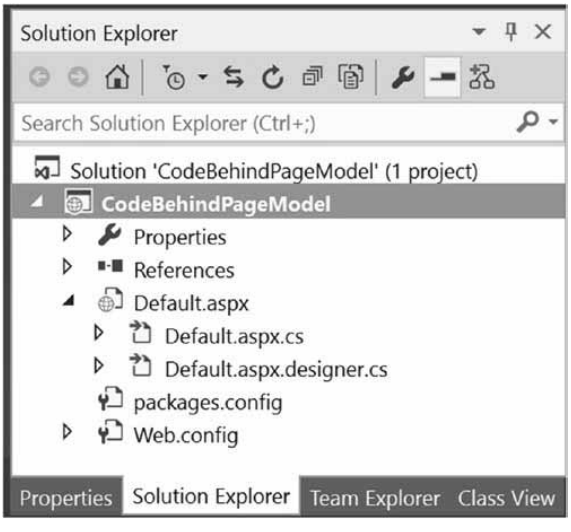



**Рис. B.16.** Файл отделенного кода, связанный с заданным файлом `*.aspx`

В файле отделенного кода обнаружится частичный класс, производный от `System.Web.UI.Page`, с поддержкой обработки события `Load`. Обратите внимание, что полностью заданное имя этого класса (`CodeBehindPageModel.Default`) идентично имени, указанному в атрибуте `Inherits` директивы `<%@ Page %>`:

```
public partial class Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
  }
}
```

## Ссылка на проект AutoLotDAL

Нам потребуется модифицировать проект `AutoLotDAL` (или задействовать скомпилированную сборку `AutoLotDAL.dll`) из предыдущего примера. В случае добавления проекта `AutoLotDAL` к решению придется добавить в веб-проект ссылку на `AutoLotDAL`. Если нужно сослаться на скомпилированную сборку `AutoLotDAL.dll`, тогда необходимо добавить данный файл в папку `\bin` в окне Solution Explorer, как показано на рис. B.17 (может понадобиться щелчок на кнопке Show All Files (Показать все файлы)).

Как и в предыдущем примере, добавим в веб-проект инфраструктуру Entity Framework, щелкнув правой кнопкой мыши на имени проекта, выбрав в контекстном меню пункт Manage NuGet Packages (Управление пакетами NuGet) и установив EF. Наконец, скопируем узел `<connectionStrings>` в файл `Web.config`.

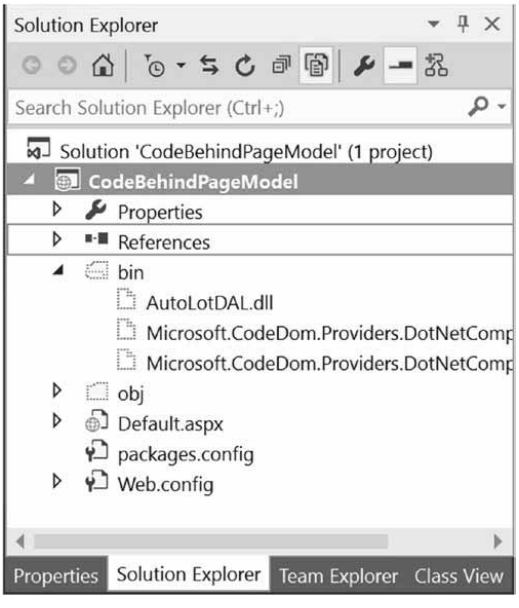



**Рис. B.17.** В веб-проектах Visual Studio используются специальные папки ASP.NET

## Обновление файла кода

Заглянув внутрь файла `Default.aspx` из предыдущего примера, можно увидеть, что каждая страница Web Forms состоит из трех файлов: файла `*.aspx` (для разметки), файла `*.designer.cs` (для кода C#, генерируемого визуальным конструктором) и главного файла кода C# (для обработчиков событий, специальных методов и т.п.), как иллюстрируется на рис. B.18.

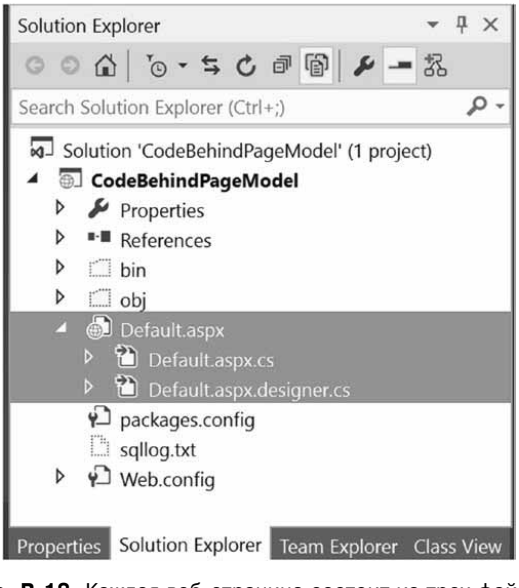



**Рис. B.18.** Каждая веб-страница состоит из трех файлов

После копирования разметки из предыдущего примера остается лишь создать метод `GetData()` в файле отделенного кода `Default.aspx.cs`. Для начала добавим операторы `using` для пространств имен `AutoLotDAL.Models` и `AutoLotDAL.Repos`. Затем добавим метод `GetData()` со следующим кодом:

```
using AutoLotDAL.Models;
using AutoLotDAL.Repos;

namespace CodeBehindPageModel
{
  public partial class Default : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    public IEnumerable<Inventory> GetData()
    {
      return new InventoryRepo().GetAll();
    }
  }
}
```

Теперь веб-приложение можно запустить, нажав комбинацию клавиш <Ctrl+F5>. Снова запустится веб-сервер IIS Express и обслужит страницу для запрашивающего браузера.

## Отладка и трассировка страниц ASP.NET

Для отладки приложений Web Forms сайт должен содержать правильно сконфигурированный файл `Web.config`. Когда запускается сеанс отладки и IDE-среда запрашивает о необходимости изменения файла `Web.config` для включения отладки, нужно ответить утвердительно. Это значит, что в файле `Web.config` отсутствует приведенная ниже разметка (наиболее важен здесь атрибут `debug="true"`):

```
<compilation debug="true" targetFramework="4.6"/>
```

Кроме того, можно включить *поддержку трассировки* для файла `.aspx`, установив атрибут `Trace` в `true` внутри директивы `<%@ Page %>` (также допускается включить трассировку для всего сайта, модифицировав файл `Web.config`):

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="CodeBehindPageModel.Default"
    Trace="true" %>
```

---

**На заметку!** Страницы веб-приложения унаследованы от класса, указанного с помощью полностью заданного имени, `CodeBehindPageModel.Default` в рассматриваемом примере. Страницы веб-сайта унаследованы от класса с именем страницы, предваренным символом подчеркивания, таким как `_Default`.

---

В результате выпущенная разметка HTML будет содержать многочисленные детали, касающиеся предыдущего цикла “запрос/ответ” HTTP (серверные переменные, переменные сеанса и приложения, запрос/ответ и т.д.). Для вставки собственных сообщений трассировки можно применять свойство `Trace`, унаследованное от класса `System.Web.UI.Page`. Всякий раз, когда нужно занести в журнал специальное сообщение (из блока сценария или файла исходного кода C#), необходимо просто вызывать статический метод `Trace.Write()`. Первый аргумент представляет имя специальной

категории, а второй — сообщение трассировки. В целях иллюстрации изменим код метода `GetData()`, как показано ниже:

```
public IEnumerable<Inventory> GetData()
{
  Trace.Write("Default.aspx","Getting Data");
  return new InventoryRepo().GetAll();
}
```

Запустите проект снова. В журнале появится специальная категория и специальное сообщение. На рис. В.19 обратите внимание на выделенное сообщение с информацией трассировки.

**Trace Information**

| Category | Message | From First(s) | From Last(s) |
|---|---|---|---|
| aspx.page | Begin PreInit | | |
| aspx.page | End PreInit | 0.000054 | 0.000054 |
| aspx.page | Begin Init | 0.000083 | 0.000029 |
| aspx.page | End Init | 0.035438 | 0.035356 |
| aspx.page | Begin InitComplete | 0.035471 | 0.000032 |
| aspx.page | End InitComplete | 0.035483 | 0.000012 |
| aspx.page | Begin PreLoad | 0.035494 | 0.000011 |
| aspx.page | End PreLoad | 0.035515 | 0.000021 |
| aspx.page | Begin Load | 0.035525 | 0.000010 |
| aspx.page | End Load | 0.035875 | 0.000350 |
| aspx.page | Begin LoadComplete | 0.035890 | 0.000015 |
| aspx.page | End LoadComplete | 0.041524 | 0.005634 |
| aspx.page | Begin PreRender | 0.041583 | 0.000059 |
| Default.aspx | Getting Data | 0.071696 | 0.030112 |
| aspx.page | End PreRender | 0.732762 | 0.661067 |
| aspx.page | Begin PreRenderComplete | 0.732802 | 0.000040 |
| aspx.page | End PreRenderComplete | 0.732812 | 0.000010 |
| aspx.page | Begin SaveState | 0.735502 | 0.002690 |
| aspx.page | End SaveState | 0.748051 | 0.012549 |
| aspx.page | Begin SaveStateComplete | 0.748073 | 0.000022 |
| aspx.page | End SaveStateComplete | 0.821923 | 0.073850 |
| aspx.page | Begin Render | 0.821966 | 0.000043 |
| aspx.page | End Render | 0.826076 | 0.004110 |

**Рис. В.19.** Занесение в журнал специальных сообщений трассировки

Теперь вы знаете, как строить страницу Web Forms с использованием однофайлового подхода и подхода с отделенным кодом. Оставшийся материал настоящего приложения посвящен анализу состава проекта Web Forms, а также способам взаимодействия с запросами/ответами HTTP и жизненному циклу класса, производного от `Page`.

---

**Исходный код.** Веб-сайт `CodeBehindPageModel` доступен в подкаталоге `Appendix_C`.

---

# Сравнение веб-сайтов и веб-приложений ASP.NET

Перед построением нового проекта Web Forms необходимо выбрать один из двух форматов проектов: *веб-сайт ASP.NET* или *веб-приложение ASP.NET*. Такой выбор будет определять то, каким образом Visual Studio организует и обрабатывает стартовые файлы веб-приложения, тип создаваемых начальных файлов проекта и степень контроля над результирующим составом скомпилированной сборки .NET.

Когда инфраструктура ASP.NET впервые появилась в составе платформы .NET 1.0, единственным вариантом было построение того, что сейчас называется *веб-приложением*. В рамках такой модели вы имеете непосредственный контроль над именем и местоположением скомпилированной выходной сборки.

Веб-приложения удобны, когда нужно переносить старые веб-сайты .NET 1.1 в проекты .NET 2.0 и последующих версий. Они также полезны, если требуется создать единственное решение Visual Studio, которое может содержать несколько проектов (например, веб-приложение и любые связанные библиотеки кода .NET). В предыдущих двух примерах в качестве отправной точки применялись веб-приложения ASP.NET.

---

**На заметку!** Из-за того, что шаблоны проектов ASP.NET в Visual Studio могут генерировать значительный объем стартового кода (мастер-страницы, страницы содержимого, библиотеки сценариев, страница входа и т.д.), в книге будет использоваться только шаблон пустого веб-сайта. Тем не менее, когда вы закончите чтение приложений, посвященных ASP.NET, обязательно создайте новый проект веб-сайта ASP.NET и просмотрите стартовый код.

---

По разительному контрасту шаблоны проектов веб-сайтов ASP.NET в Visual Studio (доступные через пункт меню File⇨New Web Site (Файл⇨Создать веб-сайт)) скрывают файл `.designer.cs` в пользу находящегося в памяти частичного класса. Более того, проекты веб-сайтов ASP.NET поддерживают несколько папок со специальными именами, такими как `App_Code`. В эту папку можно помещать любые файлы кода C# (или VB), которые напрямую не отображаются на веб-страницы, и компилятор времени выполнения по мере необходимости будет их динамически компилировать. В итоге получается значительное упрощение обычного процесса построения выделенной библиотеки кода .NET и ссылки на нее в новых проектах.

К слову, проект веб-сайта можно перенести в неизменном виде на производственный веб-сервер без предварительной компиляции сайта, которую требуется делать в случае веб-приложения ASP.NET.

В приводимых примерах применяются типы проектов веб-сайтов ASP.NET, поскольку они упрощают процесс построения веб-приложений на платформе .NET. Однако независимо от выбранного подхода вы будете иметь доступ к той же самой модели программирования.

### Включение поддержки C# 6 для веб-сайтов ASP.NET

По умолчанию поддержка C# 6 для веб-сайтов ASP.NET не включена (для проектов Web Forms она включена в стандартном шаблоне проекта). Для включения поддержки новых языковых средств C# 6 понадобится установить NuGet-пакет CodeDOM Providers for .NET Compiler Platform ("Roslyn") (Поставщики CodeDOM для платформы компилятора .NET ("Roslyn")). Чтобы установить его для веб-сайтов, щелкнем правой кнопкой мыши на имени сайта в окне Solution Explorer, выберем в контекстном меню пункт Manage NuGet Packages (Управление пакетами NuGet) и выполним поиск по строке `CodeDom`. В результате отобразится пакет `Microsoft.CodeDom.Providers.DotNetCompilerPlatform`. Щелкнем на кнопке Install (Установить).

## Структура каталогов веб-сайта ASP.NET

Когда создается новый проект веб-сайта ASP.NET, он может содержать несколько специфически именованных подкаталогов, каждый из которых имеет специальное значение для исполняющей среды ASP.NET. Такие специальные подкаталоги кратко описаны в табл. B.2.

**Таблица B.2. Специальные подкаталоги ASP.NET**

| Подкаталог | Описание |
|---|---|
| App_Browsers | Папка для файлов определений браузеров, используемых для идентификации индивидуальных браузеров и выяснения их возможностей |
| App_Code | Папка для исходного кода компонентов или классов, которые должны компилироваться как составные части приложения. Исполняющая среда ASP.NET компилирует код в этой папке при запросе страниц. Код в папке App_Code автоматически доступен приложению |
| App_Data | Папка для хранения файлов Access (*.mdb), файлов SQL Express (*.mdf), файлов XML или других хранилищ данных |
| App_GlobalResources | Папка для файлов *.resx, доступных программно из кода приложения |
| App_LocalResources | Папка для файлов *.resx, привязанных к определенной странице |
| App_Themes | Папка, содержащая коллекцию файлов, которые определяют внешний вид страниц и элементов управления Web Forms |
| App_WebReferences | Папка для прокси-классов, схем и других файлов, связанных с использованием веб-служб в приложении |
| Bin | Папка для скомпилированных закрытых сборок (файлов .dll). Сборки из папки Bin автоматически доступны приложению |

Любой из перечисленных в табл. B.2 известных подкаталогов можно явно добавить в текущее веб-приложение, выбрав пункт меню Website⇨Add ASP.NET Folder (Веб-сайт⇨Добавить папку ASP.NET). Тем не менее, во многих случаях IDE-среда будет делать это автоматически при естественном добавлении связанных файлов к сайту. Например, вставка нового файла класса в проект автоматически добавит в структуру каталогов папку App_Code, если она еще не существует.

## Ссылка на сборки

Несмотря на то что шаблоны веб-сайтов генерируют файл .sln для загрузки файлов .aspx в IDE-среду, файла .csproj больше нет. Однако проекты веб-приложений ASP.NET записывают все внешние сборки в файл .csproj. Тогда где записываются внешние сборки в ASP.NET?

Вы уже видели, что при ссылке на закрытую сборку Visual Studio автоматически создает внутри структуры каталогов подкаталог \bin для хранения локальной копии данной двоичной сборки. Когда кодовая база задействует типы из библиотеки кода, она автоматически загружается по требованию.

Если производится ссылка на разделяемую сборку, находящуюся в глобальном кеше сборок (Global Assembly Cache — GAC), то среда Visual Studio автоматически вставляет в текущее веб-решение файл Web.config (при его отсутствии) и записывает внешнюю ссылку в элемент <assemblies>. Например, если выбрать пункт меню Website⇨Add Reference (Веб-сайт⇨Добавить ссылку) и указать разделяемую сборку (такую как System.Security.dll), тогда файл Web.config обновится следующим образом:

```
<assemblies>
  <add assembly="System.Security, Version=4.0.0.0,
       Culture=neutral, PublicKeyToken=B03F5F7F11D50A3A"/>
</assemblies>
```

Итак, каждая сборка описывается с применением одной и той же информации, требуемой для динамической загрузки посредством метода Assembly.Load() (см. главу 15).

### Роль папки App_Code

Папка App_Code используется для хранения файлов кода, которые напрямую не привязаны к конкретной веб-странице (подобно файлу отделенного кода), но должны быть скомпилированы для потребления веб-сайтом. Код внутри папки App_Code будет автоматически компилироваться на лету по мере необходимости. Затем сборка станет доступной любому другому коду в рамках веб-сайта. В данном отношении папка App_Code очень похожа на папку Bin, но только в ней можно хранить исходный код, а не скомпилированный. Основное преимущество такого подхода в том, что появляется возможность определять специальные типы для веб-приложения, не компилируя их самостоятельно.

Единственная папка App_Code может содержать код на разных языках программирования. Во время выполнения для генерации сборки вызывается подходящий компилятор. Тем не менее, если предпочтительнее разбить код на части, тогда можно определить несколько подкаталогов для хранения любого количества файлов управляемого кода (*.vb, *.cs и т.д.).

Например, предположим, что мы добавили в корневой каталог приложения вебсайта папку App_Code с двумя подпапками, MyCSharpCode и MyVbNetCode, которые содержат файлы, специфичные для языка программирования. Затем мы можем обновить файл Web.config, чтобы указать эти подкаталоги с использованием элемента <codeSubDirectories>, вложенного внутрь элемента <configuration>:

```
<compilation debug="true" strict="false" explicit="true">
  <codeSubDirectories>
    <add directoryName="MyCSharpCode" />
    <add directoryName="MyVbNetCode" />
  </codeSubDirectories>
</compilation>
```

**На заметку!** В папке App_Code также будут храниться файлы, которые не являются языковыми, но необходимы для других целей (*.xsd, *.wsdl и т.д.).

Помимо Bin и App_Code существуют еще два дополнительных специальных подкаталога, App_Data и App_Themes, с которыми вы должны быть знакомы; оба они рассматриваются в последующих приложениях. Как всегда, за подробной информацией относительно оставшихся подкаталогов ASP.NET обращайтесь в документацию .NET Framework 4.7 SDK.

## Цепочка наследования типа Page

Все веб-страницы .NET в конечном итоге унаследованы от класса System.Web.UI.Page. Подобно любому базовому классу тип Page предоставляет полиморфный интерфейс для всех производных типов. Однако тип Page — не единственный член иерархии наследования. Если найти в браузере объектов Visual Studio класс System.Web.UI.Page (внутри сборки System.Web.dll), то можно обнаружить, что Page "является" классом TemplateControl, который в свою очередь "является" классом Control, а тот — классом Object (рис. B.20).

Каждый из этих базовых классов привносит изрядную часть функциональности во все файлы *.aspx. В большинстве проектов будут применяться члены, определенные внутри родительских классов Page и Control. Функциональность, полученная от класса System.Web.UI.TemplateControl, интересна только при построении специальных элементов управления Web Forms или при взаимодействии с процессом визуализации.

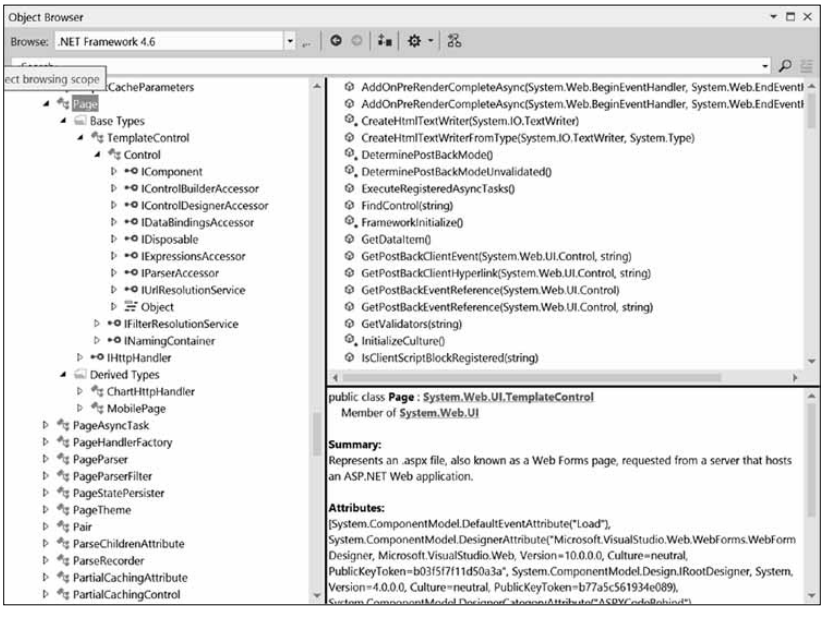



**Рис. В.20.** Цепочка наследования класса `Page`

Первый родительский класс, который мы рассмотрим — собственно `Page`. Он содержит многочисленные свойства, которые позволяют взаимодействовать с разнообразными веб-примитивами, такими как переменные приложения и сеанса, поддержка цикла "запрос/ответ" HTTP и т.д. В табл. В.3 кратко описаны некоторые (но, конечно же, не все) основные свойства.

**Таблица В.3. Избранные свойства типа `Page`**

| Свойство | Описание |
|---|---|
| `Application` | Позволяет взаимодействовать с данными, которые доступны по всему веб-сайту всем пользователям |
| `Cache` | Позволяет взаимодействовать с объектом кеша для текущего веб-сайта |
| `ClientTarget` | Позволяет указать, как данная страница должна визуализироваться в запрашивающем браузере |
| `IsPostBack` | Получает значение, которое указывает, загружается страница в ответ на клиентскую обратную отправку или при обращении к ней в первый раз |
| `MasterPageFile` | Устанавливает мастер-страницу для текущей страницы |
| `Request` | Предоставляет доступ к текущему HTTP-запросу |
| `Response` | Позволяет взаимодействовать с исходящим HTTP-ответом |
| `Server` | Предоставляет доступ к объекту `HttpServerUtility`, который содержит различные вспомогательные функции серверной стороны |
| `Session` | Позволяет взаимодействовать с данными сеанса для текущего вызывающего компонента |
| `Theme` | Получает или устанавливает имя темы, используемой для текущей страницы |
| `Trace` | Предоставляет доступ к объекту `TraceContext`, который позволяет заносить в журнал специальные сообщения во время сеансов отладки |

# Взаимодействие с входящим HTTP-запросом

Ранее в приложении вы узнали, что базовый поток веб-приложения начинается с запроса веб-страницы клиентом, возможного ввода пользователем информации и щелчка на "кнопке отправки" для обратной отправки данных формы HTML указанной веб-странице на обработку. В большинстве случаев в открывающем дескрипторе `<form>` присутствуют атрибуты `action` и `method`, указывающие файл на веб-сервере, которому будут отправлены данные из разнообразных виджетов HTML, и метод отправки данных (`GET` или `POST`):

```
<form name="defaultPage" id="defaultPage"
      action="http://localhost/Cars/ClassicAspPage.asp" method = "GET">
  ...
</form>
```

Все страницы ASP.NET поддерживают свойство `System.Web.UI.Page.Request`, которое обеспечивает доступ к экземпляру класса `HttpRequest` (основные его члены перечислены в табл. B.4).

**Таблица B.4. Члены класса `HttpRequest`**

| Член | Описание |
|---|---|
| `ApplicationPath` | Получает виртуальный корневой путь к приложению ASP.NET на сервере |
| `Browser` | Предоставляет информацию о возможностях клиентского браузера |
| `Cookies` | Получает коллекцию cookie-наборов, отправленную клиентским браузером |
| `FilePath` | Указывает виртуальный путь текущего запроса |
| `Form` | Получает коллекцию переменных формы HTTP |
| `Headers` | Получает коллекцию заголовков HTTP |
| `HttpMethod` | Указывает метод передачи данных HTTP, применяемый клиентом (`GET`, `POST`) |
| `IsSecureConnection` | Указывает, является ли подключение HTTP защищенным (т.е. HTTPS) |
| `QueryString` | Получает коллекцию переменных строки HTTP-запроса |
| `RawUrl` | Получает низкоуровневый URL текущего запроса |
| `RequestType` | Указывает метод передачи данных HTTP, используемый клиентом (`GET`, `POST`) |
| `ServerVariables` | Получает коллекцию переменных веб-сервера |
| `UserHostAddress` | Получает IP-адрес хоста удаленного клиента |
| `UserHostName` | Получает имя DNS удаленного клиента |

В дополнение к указанным в табл. B.4 членам класс `HttpRequest` содержит несколько полезных методов, включая описанные ниже.

- `MapPath()`. Отображает виртуальный путь в запрошенном URL на физический путь на сервере для текущего запроса.
- `SaveAs()`. Сохраняет информацию о текущем HTTP-запросе в файле на веб-сервере, что может оказаться полезным для целей отладки.
- `ValidateInput()`. Если включено средство проверки достоверности посредством атрибута `Validate` директивы `Page`, то этот метод можно вызывать для проверки всех введенных пользователем данных (включая cookie-наборы) по заранее определенному списку потенциально опасных входных данных.

## Получение статистики о браузере

Первый интересный аспект типа `HttpRequest` связан со свойством `Browser`, которое предоставляет доступ к лежащему в основе объекту `HttpBrowserCapabilities`. В свою очередь объект `HttpBrowserCapabilities` содержит многочисленные члены, позволяющие программно исследовать статистику относительно браузера, который отправил входящий HTTP-запрос.

Создадим новый проект пустого веб-сайта ASP.NET (по имени `FunWithPageMembers`), выбрав пункт меню File⇨New Web Site (Файл⇨Создать веб-сайт). Диалоговое окно New Web Site (Новый веб-сайт) будет выглядеть так, как показано на рис. B.21.

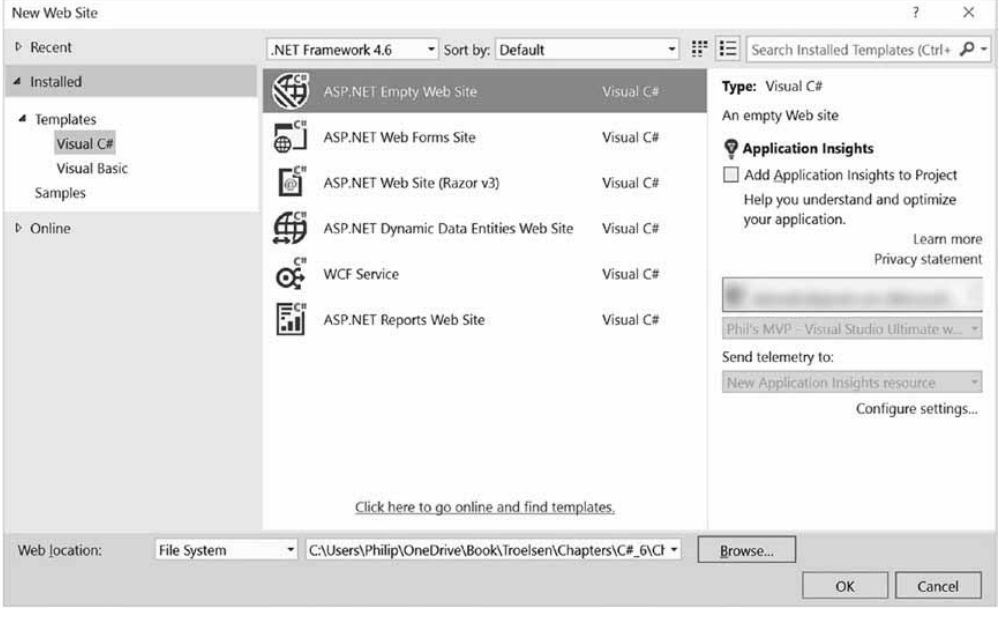



**Рис. B.21.** Создание нового пустого веб-сайта

Обратите внимание на рис. B.21, что есть возможность задать местоположение для нового веб-сайта. Если в раскрывающемся списке выбрать вариант File System (Файловая система), тогда файлы содержимого будут помещены в локальный каталог, а страницы будут обслуживаться веб-сервером IIS Express. В случае выбора варианта FTP или HTTP сайт будет размещен в новом виртуальном каталоге, поддерживаемом IIS.

В рассматриваемом примере не имеет значения, какой вариант будет выбран, но для простоты выберем File System.

Когда выбирается каталог, в котором уже имеется какой-то веб-сайт (или по существу любые файлы), в открывшемся диалоговом окне, показанном на рис. B.22, будет предложено ввести новое имя (что приведет к созданию нового каталога).

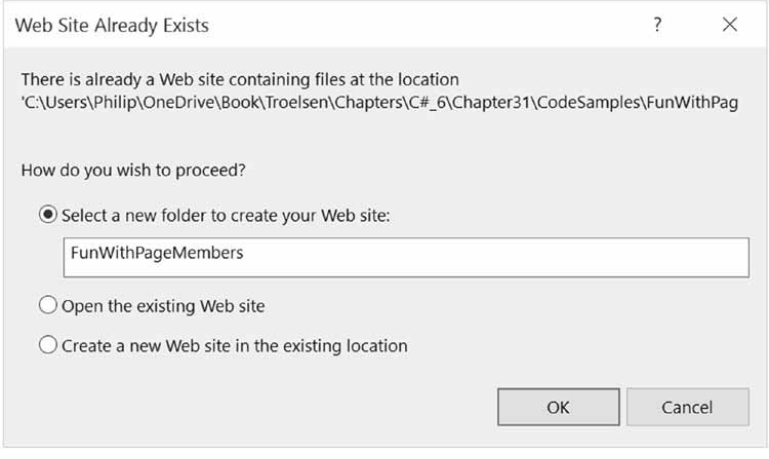



**Рис. B.22.** Указание имени для нового пустого веб-сайта

Создав пустой веб-сайт, добавим в проект новый файл Web Forms через пункт меню Website⇨Add New Item (Веб-сайт⇨Добавить новый элемент). Выберем Visual C# в древовидном представлении слева и назначим файлу имя `Default.aspx`. Первая задача заключается в построении пользовательского интерфейса, который позволит пользователям щелкать на веб-элементе управления `Button` (по имени `btnGetBrowserStats`) для просмотра разнообразных статистических данных о вызывающем браузере. Статистические сведения будут генерироваться динамическим образом и затем присоединяться к элементу управления `Label` (с именем `lblOutput`). Поместим упомянутые два элемента управления в любое место графического конструктора веб-страницы. После этого обработаем событие `Click` для кнопки, добавив к ее дескриптору атрибут `OnClick` и указав метод `btnGetBrowserStats()`. Вот разметка:

```
<strong style="font-weight: 700">Basic Request / Response Info<br />
<br />
<asp:Button ID="btnGetBrowserStats" runat="server"
     OnClick="btnGetBrowserStats_Click" Text="Get Stats" />
<br />
<br />
<asp:Label ID="lblOutput" runat="server"></asp:Label>
</strong>
```

В файле отделенного кода для страницы Web Forms реализуем обработчик, как показано далее (обратите внимание на применение интерполяции строк, которая объяснялась в главе 3):

```
protected void btnGetBrowserStats_Click(object sender, EventArgs e)
{
  string theInfo = "";
  // Клиент AOL?
  theInfo += $"<li>Is the client AOL? {Request.Browser.AOL}</li>";
```

```
    // Поддерживает ли ActiveX?
    theInfo += $"<li>Does the client support ActiveX?
{Request.Browser.ActiveXControls}</li>";
    // Бета-версия?
    theInfo += $"<li>Is the client a Beta? {Request.Browser.Beta}</li>";
    // Поддерживает ли Java-аплеты?
    theInfo += $"<li>Does the client support Java Applets?
{Request.Browser.JavaApplets}</li>";
    // Поддерживает ли cookie-наборы?
    theInfo += $"<li>Does the client support Cookies?
{Request.Browser.Cookies}</li>";
    // Поддерживает ли VBScript?
    theInfo += $"<li>Does the client support VBScript? {Request.Browser.VBScript}</li>";
    lblOutput.Text = theInfo;
   }
```

Здесь производится проверка ряда возможностей браузера. Как и можно было предположить, выяснять наличие поддержки браузером элементов ActiveX, Java-аплетов и кода VBScript клиентской стороны очень полезно. Если вызывающий браузер не поддерживает какую-то веб-технологию, то страница `.aspx` сможет предпринять альтернативные действия.

## Доступ к входным данным формы

В классе `HttpResponse` также определены свойства `Form` и `QueryString`. Они позволяют просматривать входные данные формы в виде пар "имя-значение". Хотя свойства `HttpRequest.Form` и `HttpRequest.QueryString` можно было бы использовать для доступа к данным формы, предоставленным клиентом, на веб-сервере, ASP.NET предлагает более элегантный объектно-ориентированный подход. Учитывая, что ASP.NET снабжает вас веб-элементами управления серверной стороны, элементы пользовательского интерфейса HTML можно трактовать как настоящие объекты. Следовательно, вместо того чтобы получать значение из текстового поля как в приведенном далее коде:

```
protected void btnGetFormData_Click(object sender, System.EventArgs e)
{
  // Получить значение из виджета с идентификатором txtFirstName.
  string firstName = Request.Form("txtFirstName");
  // Использовать полученное значение на странице...
}
```

можно просто обратиться к виджету серверной стороны напрямую через свойство `Text`:

```
protected void btnGetFormData_Click(object sender, System.EventArgs e)
{
  // Получить значение из виджета с идентификатором txtFirstName.
  string firstName = txtFirstName.Text;
  // Использовать полученное значение на странице...
}
```

В целях иллюстрации добавим к форме элементы управления `TextBox` и `Button`. Установим атрибут `Id` для `TextBox` в `txtFirstName`, а `Id` для `Button` — в `btnGetFormData`. Добавим атрибут `OnClick` и установим его в `btnGetFormData_OnClick`:

```
<br/>
<label>First Name</label>
<asp:TextBox runat="server" Id="txtFirstName"/>
<asp:Button runat="server" Id="btnGetFormData"
            OnClick="btnGetFormData_Click" Text="Get First Name"/>
```

Реализуем обработчик события `btnGetFormData_Click()`, как было показано выше. Запустим приложение, введем имя в элементе `TextBox` и щелкнем на кнопке Get First Name (Получить имя). Введенное имя отобразится в элементе `Label`.

Такой подход не только соответствует основополагающим принципам объектно-ориентированного программирования, но также позволяет не беспокоиться о способе отправки данных формы (`GET` или `POST`) перед получением значений. Более того, работа с виджетом напрямую намного безопаснее в отношении типов, поскольку ошибки обнаруживаются на этапе компиляции, а не во время выполнения. Конечно, речь не идет о том, что вы *никогда* не будете работать со свойствами `Form` и `QueryString` в ASP.NET; просто потребность в них значительно снижена, поэтому применять их необязательно.

### Свойство `IsPostBack`

Еще одним очень важным членом класса `Page` является свойство `IsPostBack`. Вспомните, что понятием "обратная отправка" обозначается ситуация, когда веб-страница отправляет данные обратно по тому же самому URL на веб-сервере. С учетом такого определения запомните, что свойство `IsPostBack` возвратит `true`, если текущий HTTP-запрос был отправлен пользователем текущего сеанса, и `false`, если происходит первое взаимодействие пользователя со страницей.

Необходимость в выяснении, является ли текущий HTTP-запрос на самом деле обратной отправкой, обычно возникает, когда некоторый блок кода должен выполняться только в случае, если пользователь впервые обращается к заданной странице. Например, когда пользователь получает доступ к файлу `.aspx` в первый раз, можно заполнить данными объект `DataSet` из ADO.NET и кешировать его для последующего использования. Тогда при возвращении пользователя на страницу удастся избежать излишнего обращения к базе данных (разумеется, некоторые страницы могут требовать обновления объекта `DataSet` в каждом запросе, но это совсем другая проблема). Предполагая, что страница `.aspx` обрабатывает событие `Load` (объясняется далее в приложении), запрограммировать проверку условия обратной отправки можно следующим образом:

```
protected void Page_Load(object sender, EventArgs e)
{
  // Заполнить DataSet только при самом первом
  // входе пользователя на страницу.
  if (!IsPostBack)
  {
    // Заполнить объект DataSet и кешировать его.
  }
  // Использовать кешированный объект DataSet.
}
```

## Взаимодействие с исходящим HTTP-ответом

Теперь, когда вы лучше понимаете, как тип `Page` позволяет взаимодействовать с входящим HTTP-запросом, необходимо научиться взаимодействовать с исходящим HTTP-ответом. В ASP.NET свойство `Response` класса `Page` предоставляет доступ к экземпляру типа `HttpResponse`. В типе `HttpResponse` определен набор свойств, которые позволяют форматировать HTTP-ответ, отправляемый обратно клиентскому браузеру. Основные свойства `HttpResponse` перечислены в табл. B.5.

**Таблица B.5. Свойства типа `HttpResponse`**

| Свойство | Описание |
|---|---|
| `Cache` | Возвращает семантику кеширования веб-страницы (см. приложение Д) |
| `ContentEncoding` | Получает или устанавливает набор символов HTTP для выходного потока |
| `ContentType` | Получает или устанавливает MIME-тип HTTP для выходного потока |
| `Cookies` | Получает коллекцию `HttpCookie`, которая будет возвращена браузеру |
| `Output` | Позволяет осуществлять текстовый вывод в тело исходящего HTTP-ответа |
| `OutputStream` | Позволяет осуществлять двоичный вывод в тело исходящего HTTP-ответа |
| `StatusCode` | Получает или устанавливает код состояния HTTP выходных данных, возвращаемых клиенту |
| `StatusDescription` | Получает или устанавливает строку состояния HTTP выходных данных, возвращаемых клиенту |
| `SuppressContent` | Получает или устанавливает значение, которое указывает, что HTTP-ответ не будет отправлен клиенту |

В табл. B.6 приведен частичный список методов, поддерживаемых типом `HttpResponse`.

**Таблица B.6. Методы типа `HttpResponse`**

| Метод | Описание |
|---|---|
| `Clear()` | Очищает все заголовки и выходное содержимое из буфера потока |
| `End()` | Отправляет весь буферизованный вывод клиенту, после чего закрывает сокетное подключение |
| `Flush()` | Отправляет весь текущий буферизованный вывод клиенту |
| `Redirect()` | Перенаправляет клиента на новый URL |
| `Write()` | Записывает значения в выходной поток содержимого HTTP |
| `WriteFile()` | Записывает файл непосредственно в выходной поток содержимого HTTP |

## Выпуск содержимого HTML

Вероятно, самым известным аспектом типа `HttpResponse` является способность записывать содержимое прямо в выходной поток HTTP. Метод `HttpResponse.Write()` позволяет передавать в поток любые дескрипторы HTML и/или текстовые литералы. Метод `HttpResponse.WriteFile()` еще больше развивает эту функциональность, позволяя указывать имя физического файла на веб-сервере, содержимое которого должно быть помещено в выходной поток (что очень удобно для быстрого выпуска содержимого существующего файла `.html`).

В целях демонстрации добавим в текущий файл `.aspx` еще один элемент управления `Button`:

```
<br/>
<asp:Button runat="server" Id="btnHttpResponse"
   OnClick="btnHttpResponse_Click" Text="Get First Name"/>
```

Реализуем обработчик события `Click` серверной стороны:

```
protected void btnHttpResponse_Click(object sender, EventArgs e)
{
  Response.Write("<b>My name is:</b><br>");
  Response.Write(this.ToString());
}
```

Роль этого вспомогательного метода (который, как и можно было предположить, вызывается некоторыми обработчиками событий на стороне сервера) довольно проста. Хотя всегда можно прибегнуть к старому подходу и генерировать дескрипторы HTML и содержимое с применением метода `Write()`, в ASP.NET такой способ встречается намного реже, чем в классическом ASP. Причина (опять-таки) связана с появлением веб-элементов управления серверной стороны. Таким образом, чтобы визуализировать блок текстовых данных в браузере, достаточно присвоить нужную строку свойству `Text` виджета `Label`.

### Перенаправление пользователей

Еще одним аспектом класса `HttpResponse` является способность перенаправления пользователей на новый URL, например:

```
protected void btnWasteTime_Click(object sender, EventArgs e)
{
  Response.Redirect("http://www.facebook.com");
}
```

Если обработчик событий `btnWasteTime_Click()` вызывается через обратную отправку клиентской стороны, тогда пользователь будет автоматически перенаправлен на указанный URL.

---

**На заметку!** Метод `HttpResponse.Redirect()` всегда влечет за собой взаимодействие с клиентским браузером. Если нужно просто передать управление файлу `.aspx` из того же самого виртуального каталога, то более эффективно использовать метод `HttpServerUtility.Transfer()`, доступный через унаследованное свойство `Server`.

---

Представленного материала вполне достаточно для оценки функциональности класса `System.Web.UI.Page`. Роль базового класса `System.Web.UI.Control` будет исследоваться в следующем приложении. А теперь давайте рассмотрим жизненный цикл объекта, производного от `Page`.

---

**Исходный код.** Веб-сайт `FunWithPageMembers` доступен в подкаталоге `Appendix_C`.

---

## Жизненный цикл страницы Web Forms

Каждая страница Web Forms имеет фиксированный жизненный цикл. Когда исполняющая среда ASP.NET принимает входящий запрос некоторого файла `.aspx`, в памяти размещается соответствующий объект производного от `System.Web.UI.Page` типа с применением стандартного конструктора этого типа. Затем инфраструктура автоматически запускает последовательность событий. По умолчанию принимается во внимание событие `Load`, в тело обработчика которого можно поместить свой специальный код:

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
```

```
    Response.Write("Load event fired!");
  }
}
```

Кроме события `Load` заданный объект `Page` способен перехватывать любые основные события из табл. В.7, которые приведены в порядке их появления (подробные сведения о событиях, возникающих на протяжении времени жизни страницы, ищите в документации .NET Framework 4.7 SDK).

**Таблица В.7. Избранные события типа `Page`**

| Событие | Описание |
|---|---|
| `PreInit` | Инфраструктура использует это событие для размещения в памяти всех веб-элементов управления, применения тем, установки мастер-страницы и настройки пользовательских профилей. Данное событие можно перехватить, чтобы вмешаться в процесс |
| `Init` | Инфраструктура использует это событие для установки свойств веб-элементов управления в их предыдущие значения посредством обратной отправки или данных состояния представления |
| `Load` | Когда возникает это событие, страница и ее элементы управления полностью инициализированы, а их предыдущие значения восстановлены. В данный момент можно безопасно взаимодействовать с каждым веб-виджетом |
| "Событие, которое инициировало обратную отправку" | Конечно, события с таким именем нет. Подобным образом просто обозначается любое событие, из-за которого браузер выполнил обратную отправку веб-серверу (вроде щелчка на кнопке) |
| `PreRender` | Привязка данных для элементов управления и конфигурирование пользовательского интерфейса выполнены, а элементы управления готовы визуализировать свои данные в исходящий HTTP-ответ |
| `Unload` | Страница и ее элементы управления завершили процесс визуализации, и объект страницы готов к уничтожению. В данный момент попытка взаимодействия с исходящим HTTP-ответом приведет к ошибке времени выполнения. Тем не менее, это событие можно перехватывать для проведения любой очистки на уровне страницы (закрыть файл или подключение к базе данных, выполнить действие регистрации в журнале, освободить память, занимаемую объектами, и т.д.) |

Как ни странно, IDE-среда не поддерживает обработку других событий помимо `Load`. В таком случае требуется вручную писать в файле кода метод с именем `Page_ИмяСобытия`. Например, вот каким образом можно обработать событие `Unload`:

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    Response.Write("Load event fired!");
  }

  protected void Page_Unload(object sender, EventArgs e)
  {
    // Больше невозможно помещать данные в HTTP-ответ,
    // поэтому будем записывать их в локальный файл.
    System.IO.File.WriteAllText(@"C:\MyLog.txt", "Page unloading!");
  }
}
```

**На заметку!** Каждое событие, определенное типом `Page`, работает в сочетании с делегатом `System.EventHandler`; следовательно, обработчики этих событий всегда принимают `Object` в первом параметре и `EventArgs` — во втором.

## Роль атрибута `AutoEventWireup`

Чтобы организовать обработку событий для своей страницы, необходимо добавить подходящий обработчик в блок `<script>` или файл отделенного кода. Однако, исследуя директиву `<%@ Page %>`, можно заметить специфический атрибут по имени `AutoEventWireup`, который по умолчанию установлен в `true`:

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

При таком стандартном поведении каждый обработчик события уровня страницы будет автоматически добавляться при вводе метода с соответствующим именем. Если же установить атрибут `AutoPageWireup` в `false`:

```
<%@ Page Language="C#" AutoEventWireup="false"
    CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

тогда события уровня страницы перехватываться не будут. Включение атрибута `AutoPageWireup` приводит к генерации необходимой оснастки для событий внутри автоматически сгенерированного частичного класса, который был описан ранее в настоящем приложении. Даже если атрибут `AutoEventWireup` отключен, то события уровня страницы все равно можно будет обрабатывать с применением логики обработки событий C#:

```
public _Default()
{
  // Явно привязаться к событиям Load и Unload.
  this.Load += Page_Load;
  this.Unload += Page_Unload;
}
```

Как и можно было ожидать, атрибут `AutoEventWireup` обычно оставляют во включенном состоянии.

## Событие `Error`

Во время жизненного цикла страницы может произойти еще одно событие по имени `Error`. Оно возникает, когда метод унаследованного от `Page` типа инициирует исключение, которое не было явно обработано. Предположим, что имеется обработчик события `Click` для заданного элемента управления `Button` на странице, и внутри этого обработчика (`btnGetFile_Click()`) предпринимается попытка записать в HTTP-ответ содержимое локального файла.

Также предположим, что проверка на существование файла посредством стандартной структурированной обработки исключений *отсутствует*. Если поместить в стандартный конструктор обработчик события `Error`, тогда появится последний шанс справиться с проблемой на этой странице, прежде чем конечный пользователь получит невнятное сообщение об ошибке. Взгляните на следующий код:

```
public partial class _Default : System.Web.UI.Page
{
  void Page_Error(object sender, EventArgs e)
  {
```

```
    Response.Clear();
    Response.Write("I am sorry...I can't find a required file.<br>");
    Response.Write($"The error was: <b>{ Server.GetLastError().Message }</b>");
    Server.ClearError();
  }
  protected void Page_Load(object sender, EventArgs e)
  {
    Response.Write("Load event fired!");
  }
  protected void Page_Unload(object sender, EventArgs e)
  {
    // Больше невозможно помещать данные в HTTP-ответ,
    // поэтому будем записывать их в локальный файл.
    System.IO.File.WriteAllText(@"C:\MyLog.txt", "Page unloading!");
  }
  protected void btnPostback_Click(object sender, EventArgs e)
  {
    // Здесь ничего не происходит. Это нужно только
    // для обеспечения обратной отправки страницы.
  }
  protected void btnTriggerError_Click(object sender, EventArgs e)
  {
    System.IO.File.ReadAllText(@"C:\IDontExist.txt");
  }
}
```

Обратите внимание, что обработчик события `Error` начинается с очистки любого содержимого, имеющегося в HTTP-ответе, и выдачи обобщенного сообщения об ошибке. Чтобы получить доступ к конкретному объекту `System.Exception`, можно использовать метод `HttpServerUtility.GetLastError()`, доступный через унаследованное свойство `Server`:

```
Exception e = Server.GetLastError();
```

Перед выходом из обобщенного обработчика ошибок явно вызывается метод `HttpServerUtility.ClearError()` через свойство `Server`. Вызов `ClearError()` обязателен, т.к. он информирует исполняющую среду о том, что мы решили проблему вручную, и дальнейшая обработка не требуется. Если вы забудете сделать это, то конечному пользователю отобразится страница с ошибкой времени выполнения.

К настоящему моменту вы должны хорошо понимать структуру типа `Page`. При такой подготовке можно переключать внимание на элементы управления Web Forms, темы и мастер-страницы, рассматриваемые в последующих приложениях. Тем не менее, чтобы завершить текущее приложение, давайте выясним роль файла `Web.config`.

---

**Исходный код.** Веб-сайт `PageLifeCycle` доступен в подкаталоге `Appendix_C`.

---

# Роль файла `Web.config`

По умолчанию все приложения Web Forms на C#, созданные с помощью Visual Studio, автоматически снабжаются файлом `Web.config`. Однако если возникает необходимость добавить файл `Web.config` к веб-сайту вручную (например, когда вы работаете с однофайловой моделью и не создавали веб-решение), то это делается с применением пункта меню Website⇨Add New Item (Веб-сайт⇨Добавить новый элемент). В файл `Web.config`

можно добавлять настройки, которые управляют поведением веб-приложения во время выполнения.

При рассмотрении сборок .NET (в главе 14) вы узнали, что клиентские приложения могут использовать конфигурационный файл XML для инструктирования среды CLR относительно того, как она должна обрабатывать запросы привязки, проводить зондирование сборок и поддерживать другие детали времени выполнения. То же самое остается справедливым и для приложений Web Forms, но с заметным отличием в том, что веб-ориентированные конфигурационные файлы всегда именуются `Web.config` (в отличие от конфигурационных файлов `*.exe`, которые называются по имени связанной клиентской исполняемой сборки).

Полная структура файла `Web.config` довольно многословна. Тем не менее, в табл. B.8 приведен обзор наиболее интересных элементов, которые можно обнаружить в `Web.config`.

**Таблица B.8. Избранные элементы файла `Web.config`**

| Элемент | Описание |
|---|---|
| `<appSettings>` | Этот элемент служит для установления специальных пар "имя-значение", которые с помощью типа `ConfigurationManager` можно программно прочитать в память для использования страницами |
| `<authentication>` | Этот элемент, относящийся к безопасности, служит для определения режима аутентификации для данного веб-приложения |
| `<authorization>` | Этот элемент, также связанный с безопасностью, применяется для определения того, какие пользователи имеют доступ к тем или иным ресурсам на веб-сервере |
| `<connectionStrings>` | Этот элемент используется для хранения строк внешних подключений, которые применяются внутри веб-сайта |
| `<customErrors>` | Этот элемент используется для указания исполняющей среде точного способа сообщения об ошибках, возникающих во время функционирования веб-приложения |
| `<globalization>` | Этот элемент применяется для конфигурирования настроек глобализации для данного веб-приложения |
| `<namespaces>` | Этот элемент содержит список всех пространств имен, подлежащих включению, если веб-приложение было заранее скомпилировано с использованием нового инструмента командной строки `aspnet_compiler.exe` |
| `<sessionState>` | Этот элемент применяется для управления тем, как и где исполняющая среда .NET будет хранить данные состояния сеанса |
| `<trace>` | Этот элемент используется для включения (или отключения) поддержки трассировки для данного веб-приложения |

Кроме набора, представленного в табл. B.8, файл `Web.config` может содержать дополнительные элементы. Подавляющее большинство их связано с безопасностью, в то время как остальные полезны только в расширенных сценариях работы с ASP.NET, таких как создание специальных заголовков HTTP или специальных модулей HTTP (темы, которые здесь не раскрываются).

### Утилита администрирования веб-сайтов ASP.NET

Несмотря на то что содержимое файла `Web.config` всегда можно модифицировать прямо в Visual Studio, для веб-проектов Web Forms доступен удобный веб-ориентированный редактор, который позволяет графически редактировать многочисленные элементы и атрибуты файла `Web.config`. Чтобы запустить такой инструмент, понадобится выбрать пункт меню Website⇨ASP.NET Configuration (Веб-сайт⇨Конфигурация ASP.NET).

Просмотрев содержимое вкладок в верхней части страницы, вы легко заметите, что большая часть функциональности инструмента предназначена главным образом для установления настроек безопасности веб-сайта. Однако данный инструмент также делает возможным добавление настроек к элементу `<appSettings>`, определение параметров отладки и трассировки, а также установку стандартной страницы с сообщением об ошибке.

Вы увидите этот инструмент в действии, когда он будет необходим, а сейчас имейте в виду, что утилита администрирования веб-сайтов ASP.NET *не* позволяет добавлять в файл `Web.config` абсолютно все возможные настройки. Почти наверняка будут возникать ситуации, когда файл `Web.config` придется изменять вручную с применением обычного текстового редактора.

## Резюме

По сравнению с созданием традиционных настольных приложений построение веб-приложений требует иного образа мышления. В настоящем приложении вы ознакомились с рядом основных тем, включая HTML, HTTP, роль сценариев клиентской стороны и роль сценариев серверной стороны, в которых используется классический ASP. Материал приложения по большей части был посвящен исследованию архитектуры страницы ASP.NET. Вы видели, что каждый файл `*.aspx` в проекте имеет ассоциированный с ним класс, производный от `System.Web.UI.Page`. С применением такого объектно-ориентированного подхода инфраструктура ASP.NET позволяет создавать многократно используемые объектно-ориентированные системы.

После рассмотрения основной функциональности, предлагаемой цепочкой наследования страницы, в приложении было показано, как страницы в конечном итоге компилируются в допустимую сборку .NET. Приложение завершилось обсуждением роли файла `Web.config` и кратким обзором инструмента администрирования веб-сайтов ASP.NET.