2

Разработка дизайна сайта

Первым шагом при создании нового сайта является разработка визуального дизайна, подразумевающая продумывание общей схемы сайта и использование графики. Эта визуальная архитектура определяет "внешний вид и поведение" сайта с точки зрения пользователя. Сначала вы должны решить, какое впечатление вы хотите, чтобы складывалось у пользователя при работе с вашим сайтом, и только затем приступать к разработке кода, который будет обеспечивать это впечатление. Одними из основных деталей, влияющих на впечатление пользователя, являются меню и возможности навигации, наличие изображений и организация элементов на странице. Меню должны быть интуитивно понятными и подкрепляться навигационными подсказками, такими как карта сайта или "*хлебные крошки*", которые напоминают пользователю о том, где он находятся относительно сайта в целом. В этом контексте под хлебными крошками подразумевается набор небольших ссылок на странице, которые служат путеводной нитью, позволяющей пользователям возвращаться на предыдущую страницу при помощи щелчка на соответствующей нужной странице ссылке в иерархии страниц.

Прежде чем приступать к написанию какого-либо кода, вы обязательно должны познакомиться со специальными средствами, которые доступны в ASP.NET, чтобы иметь возможность воспользоваться результатами работы, проделанной разработчиками Microsoft. Хорошо продуманная с технической точки зрения архитектура увеличивает вероятность повторного использования кода и делает его более удобным в плане обслуживания. В этой главе речь пойдет об общей структуре сайта и том, как пользоваться такими мощными функциональными возможностями как мастер-страницы (master pages) и темы (themes). Мастер-страницы применяются для объединения функциональных возможностей в шаблоны, которые предоставляют общие элементы, используемые одновременно многими страницами, а темы позволяют пользователям настраивать определенные аспекты сайта для придания им уникального вида, отвечающего именно их вкусам (это средство также называется *скиннингом* (skinning)).

Задача

Многие разработчики начинают писать код, не обращая внимания на главную задачу сайта: предоставлять простое, но имеющее множество функциональных возможностей графическое приложение пользователям для работы. Создание пользовательского интерфейса кажется очень простой задачей, но если ее не выполнить должным образом, к ней придется возвращаться во время разработки кода, причем возможно даже не один раз. Каждый такой возврат и изменение фундаментальных функциональных возможностей будет требовать затраты определенных дополнительных усилий, не говоря уже о проверке взаимодействия и функционирования всех компонентов с самого начала. Что еще хуже, слишком небрежное отношение к пользовательскому интерфейсу вполне может в конечном итоге закончиться тем, что пользователи предпочтут не пользоваться данным сайтом вообще. При создании дизайна сайта должны продумываться различные элементы. Во-первых, нужно убедить себя в одной простой вещи: внешний вид *имеет* значение! Лучше даже будет повторить это несколько раз вслух. Если сайт плохо выглядит, люди могут пожалеть, что зашли сюда. Разработчики склонны уделять больше внимания сложным задачам, таким как организация исходного кода в классы и кодирование бизнес-логики — по сравнению с ними разработка косметических средств сайта не кажется такой уж важной, правильно? Нет, не правильно! Пользовательский интерфейс — это первое, что видит конечный пользователь: если он уродливый, непонятный и вообще не удобный в использовании, скорее всего, пользователь покинет этот сайт с плохим впечатлением как о самом сайте, так и о разрабатывавших его людях. И, как ни печально, это произойдет независимо от того, насколько быстрым и масштабируемым является этот сайт. Кроме того, нужно еще учесть тот факт, что не все пользователи имеют одинаковое мнение о шаблоне сайта. Некоторые пользователи могут плохо воспринимать текст на сайте с определенной цветовой схемой и отдавать предпочтение какой-нибудь другой цветовой схеме, которая может быть непонятна многим другим пользователям. Подобрать один шаблон и одну цветовую схему, которая будет нравиться всем, очень трудно. Именно поэтому на некоторых сайтах пользователям предлагается на выбор множество различных цветовых схем и шаблонов, что позволяет им настраивать пользовательский интерфейс по своему личному вкусу — и, возможно, в соответствии со своими физическими недостатками, такими как дальтонизм, например. Исследования показали, что удивительно много людей страдает от частичной "цветовой слепоты", которая не позволяет им различать определенные цвета, а это значит, что у них должна быть возможность выбирать цвета, которые они могут различать и которые при этом все равно будут довольно приятными на вид.

Выбрав схему расположения элементов и цвета, которые будут использоваться, следует удостовериться в том, что сайт будет выглядеть одинаково во всех браузерах. Пару лет назад Internet Explorer (IE) был бесспорно доминирующим браузером среди пользователей Windows, и, разрабатывая технический сайт, ориентированный на Windows-разработчиков, можно было смело предполагать, что для просмотра сайта по большей части будет использоваться браузер IE, и, следовательно, разрабатывать и тестировать его нужно было только относительно IE. Однако сегодня среди пользователей Internet все более популярным становится браузер Mozilla Firefox, который к тому же доступен и для других операционных систем, таких как Linux и Mac OS. Поскольку расчет делается не на узкий круг пользователей (т.е., не только на Windows-разработчиков, а на всех людей, которые посещают паб клиента) и посколь-

ку существуют и другие популярные браузеры помимо IE, обязательно необходимо удостовериться в том, что сайт будет работать хорошо с большинством популярных браузеров. Если не сделать этого и ориентироваться только на IE, может случиться, что пользователи Firefox, зашедшие на сайт, обнаружат сильно отличающую от того, что они ожидали, схему расположения элементов, с неудачным выравниванием, размерами и цветами, с накладывающимися друг на друга панелями и текстом — в общем, полную неразбериху. Нетрудно догадаться, что пользователь, попавший на такую нелицеприятную страницу, естественно, покинет ее, что для электронного магазина означает потерею потенциального клиента или заказчика. В худшем случае посещение такого пользователя будет представлять собой генерацию представлений страниц и, следовательно, просмотр баннеров. Поскольку терять посетителей никто не хочет, мы будем брать в расчет как Internet Explorer, так и Firefox.

Процесс проектирования уровня пользовательского интерфейса не означает только написание HTML-кода для страницы; он также подразумевает разработку навигационной системы и обеспечение для Web-мастера или администратора сайта (если не конечного пользователя) возможности легко изменять внешний вид сайта, не редактируя сами страницы содержимого (которых очень много). Совсем не помешает разработать систему, позволяющую людям легко изменять меню сайта и его внешний вид (шрифты, цвета и размер различных компонентов, из которых состоит страница), поскольку это сводит к минимуму объем работы администраторов и делает счастливыми пользователей. После того, как будет разработана домашняя станица сайта, разработка всех остальных страниц не займет много времени, потому что домашняя страница определяет схему (layout) и элементы навигации, которые будут применяться по всему сайту. А при необходимости изменить что-нибудь в схеме сайта (например, добавить новое поле опроса так, чтобы оно отображалось в правой части любой страницы), это будет очень легко сделать, если был разработан общий интерфейс, используемый совместно многими страницами. Именно поэтому, несомненно, стоит потратить дополнительное время на продумывание уровня пользовательского интерфейса, вместо того, чтобы сразу же запускать Visual Studio .NET и начинать писать код. Это поистине стратегическое решение, которое может сэкономить несколько часов или даже дней работы в будущем. Помните о том, что фундаментальные изменения, вносимые позже в процессе разработки, будут отнимать больше времени и усилий.

Проект

В этом разделе речь пойдет о том, как можно технически решить описанные в первом разделе проблемы. На практике мы будем проектировать и реализовывать следующие вещи.

- Привлекательный графический шаблон (схему), выглядящий одинаково как в Internet Explorer, так и в Firefox, и механизм для динамического применения к нему различных цветовых схем и других касающихся внешнего вида атрибутов.
- Способ, позволяющий быстро применить созданный шаблон ко всем страницам сайта, не копируя и не вставляя весь код в каждую страницу физически.
- Навигационную систему, позволяющую легко редактировать отображающиеся в меню сайта ссылки, дающую пользователям ясно понять, где они сейчас находятся на карте сайта, и разрешающую им возвращаться обратно.

- Способ, позволяющий применить ко всем страницам сайта не только общий дизайн, но и общее поведение вроде подсчета представлений страниц или применения к странице любимого стиля пользователя.

Я расскажу, как можно применять некоторые новые функциональные возможности ASP.NET 2.0 при реализации повторного использования, меню, системы навигации и опций настройки. Позже, в разделе "Решение", эти новые мощные возможности можно будет испробовать на практике!

Проектирование схемы сайта

Когда вы разрабатываете дизайн сайта, вы обычно создаете предварительную модель при помощи какого-нибудь графического приложения типа Adobe Photoshop или Jasc Paint Shop Pro, чтобы увидеть, как может выглядеть сайт в конечном итоге, прежде чем приступать к созданию определенной схемы или кода в HTML. Создав такую предварительную модель, вы можете показать ее потенциальным пользователям, специалистам по тестированию и руководителям, чтобы получить их одобрение. Это может быть простой рисунок, такой как на рис. 2.1, иллюстрирующий, как содержимое будет размещаться в разных частях страницы.

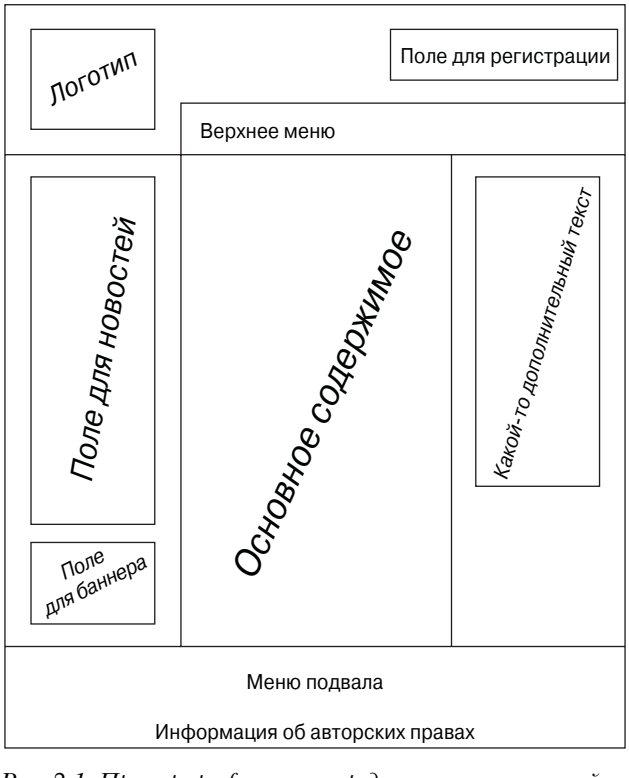

Рис. 2.1. Пример графического представления схемы сайта

Это типичная трехколоночная схема с нижним и верхним колонтитулами. После одобрения макета его следует воспроизвести с реальной графикой и HTML. Это лучше делать в графической программе, потому что у Web-разработчиков в среднем все-таки уходит гораздо меньше времени на то, чтобы воспроизвести такие предварительные модели в виде изображений, нежели на то, чтобы воспроизвести их в виде реальных HTML-страниц. Как только клиент одобрит окончательную модель, Web-дизайнер может поделить изображение модели на небольшие части и использовать их на HTML-странице.

Создание экспериментальной модели не всегда является простой задачей для тех из нас, кто не очень изобретателен по натуре. Я должен признаться, что являюсь одним из самых худших графических дизайнеров из всех, кого знаю. Для компании среднего или большого размера это не проблема, потому что в таких компаниях обычно всегда есть специальный человек, профессиональный Web-дизайнер, занимающийся разработкой графического дизайна, на основе которого разработчики (люди вроде меня или вас) затем уже создают приложение. Иногда бывает лучше воспользоваться услугами сторонней компании или, в конце концов, поручить разработку дизайна кому-нибудь другому, более одаренному в этом плане, чтобы он сделал шаблон сайта. Например, для создания Web-сайта, описываемого в этой книге, я воспользовался услугами компании TemplateMonster (www.templatemonster.com), чтобы создать для сайта привлекательный дизайн, который мог бы использовать в качестве отправной точки. Они предоставили его мне в виде PSD-файлов (которые можно открыть в Photoshop или Paint Shop Pro), JPEG-файлов и нескольких HTML-страниц с изображениями, уже порезанными на части и размещенными в нужных позициях посредством HTML-таблиц. Использовать их заготовленные HTML-страницы в таком, каком они были виде, я не мог, потому что хотел создать свои собственные стили и изменить HTML-разметку, но иметь такой наглядный шаблон для начала было даже очень полезно. Это может позволять сайту выглядеть профессионально уже на ранней стадии, благодаря чему его, естественно, будет гораздо легче продать соответствующим людям.

Технологии, необходимые для реализации дизайна

ASP.NET 2.0 — это главная технология, и именно она заставляет сайт работать. Она выполняется на Web-сервере и использует функциональные возможности, предоставляемые средой .NET Framework. Однако на компьютере пользователя ASP.NET не выполняется; вместо этого она динамически генерирует элементы, необходимые браузеру для визуализации страницы. К этим отправляемым браузеру элементам относятся HTML-код, изображения и CSS-таблицы (Cascading Style Sheets — каскадные таблицы стилей), которые предоставляют цвета, размеры и типы выравнивания для различные элементов в HTML. ASP.NET также генерирует кое-какой процедурный код JavaScript, который также отправляется браузеру, для обеспечения механизма проверки корректности данных и указания браузеру, как должно осуществляться взаимодействие с Web-сервером.

HTML-код определяется несколькими способами. Вы можете использовать визуальный конструктор форм в Visual Studio, чтобы разместить элементы управления в форме и тем самым автоматически создать HTML-код. Или же вы можете вручную отредактировать или создать свой собственный HTML-код в `.aspx`-файлах, чтобы наделить его функциональными возможностями, добавить которые в конструкторе форм очень трудно. И, наконец, HTML-код может динамически генерироваться вашим C#-кодом или классами в .NET Framework.

В ASP.NET 1.x использовалась модель отделенного кода (code-behind model): HTML-код (и кое-какой ориентированный на презентацию C#-код) помещался в файл `.aspx`, а C#-код, отвечающий за реализацию, — в отдельный файл, который наследовался от этого файла `.aspx`. Файл `.aspx` мы называем "страницей", потому что именно в нем определяется визуальная Web-страница. Это привело к тому, что между кодом представления и кодом реализации стало проводиться определенное разграничение. Одной из проблем этой модели было то, что автоматически генерируемый код, создаваемый конструктором форм, помещался в те же файлы, которые разработчик использовал для своего кода.

В ASP.NET 2.0 модель отделенного кода была изменена и теперь подразумевает использование новой функциональной возможности .NET Framework, которая называется *частичными классами* (partial classes). Идея проста: позволить одному классу охватывать более чем один файл. Visual Studio будет автоматически генерировать во время выполнения код для объявления элементов управления и регистрации событий и затем объединять его с кодом, написанным пользователем; в результате будет получаться один единственный класс, наследуемый страницей `.aspx`. Директива `@Page`, объявляемая в этой странице `.aspx`, будет содержать ссылку на файл code-behind (`.cs`) с написанным пользователем кодом в атрибуте `CodeFile`.

Еще одним изменением в ASP.NET 2.0 является исключение файлов проекта. Проекты теперь определяются на основании имеющихся на жестком диске папок. Также в ASP.NET 1.x код для всех страниц проекта генерировался в один `.dll`-файл, теперь же, в версии 2.0, код генерируется для каждой страницы отдельно. Почему это имеет значение? Да потому, что благодаря этому необходимость повторно разворачивать огромные фрагменты кода, когда изменения вносятся только в одну страницу, отпадает. Разворачивать код повторно нужно только для той страницы или страниц, которые изменяются, что дает возможность управлять изменениями на более детальном уровне.

Использование технологии CSS для определения стилей в файлах стилевых таблиц

Возможности привести исчерпывающее описание технологии CSS в этой книге у меня нет, поэтому я расскажу лишь о ее некоторых наиболее важных концепциях. Для получения полной информации о CSS вам следует пользоваться другими источниками. Задачей CSS является указывать, как должны визуализироваться визуальные HTML-дескрипторы, путем перечисления различных стилистических элементов, таких как шрифт, размер, цвет, тип выравнивания и т.п. Эти стили могут включаться в виде атрибутов HTML-дескрипторов или храниться отдельно и вызываться по имени или идентификатору.

Иногда встречаются HTML-файлы, в которых стили жестко закодированы в самих HTML-дескрипторах, как показано в следующем примере:

```
<div style="align: justify; color: red; background-color: yellow;
font-size:12px;">some text</div>
```

Ничего хорошего в этом нет, потому что изменить такие стилистические элементы, не заглянув во все HTML-файлы и не отыскав все CSS-атрибуты, очень сложно. Поэтому определения стилей лучше всегда помещать в специальный отдельный файл, называемый *файлом таблицы стилей* и имеющий расширение `.css`, или же, если стили

обязательно нужно включить в HTML-файл, тогда хотя бы в отдельный раздел, идущий в самом начале этого файла и называющийся <style>.

Группируя стили вместе, можно создавать небольшие классы, напоминающие с синтаксической точки зрения классы или функции в C#, и, следовательно, присваивать им имя класса или идентификационный номер для того, чтобы на них можно было ссылаться в атрибуте class= HTML-дескрипторов.

Если вы применяете классы таблицы стилей и хотите изменить размер шрифта всех HTML-дескрипторов, которые используют этот класс, вам нужно только отыскать объявление этого класса и внести изменение в это одно единственное вхождение: все остальные HTML-элементы данного типа будут изменены автоматически. Если таблица стилей находится в отдельном файле, вы сможете выиграть от этого подхода даже еще больше, потому что вам придется внести изменения только в один единственный файл, а внешний вид всех остальных страниц изменится соответствующим образом автоматически.

Главным преимуществом использования CSS является сведение к минимуму количества усилий администратора, необходимых для обслуживания стилей и принудительного применения определенного внешнего вида и поведения к множеству страниц. Кроме того, CSS также обеспечивает безопасность HTML-кода и сайта в целом. Давайте представим, что клиент захотел изменить какие-нибудь стили сайта, который уже используется в производственной среде. Если вы жестко закодировали стили в HTML-элементах, тогда вам придется просматривать множество файлов, чтобы найти все стили, которые нужно изменить; не исключено, что вам не удастся отыскать их все или что вы по ошибке измените какой-нибудь не тот стиль, что запросто может привести к появлению серьезной неисправности! Однако если вы использовали классы стилей, хранящиеся отдельно в специальных CSS-файлах, тогда вы сможете гораздо быстрее найти классы, в которые требуется внести изменения, и ваш HTML-код при этом останется незатронутым и в полной безопасности.

Более того, CSS-файлы могут делать сайт более эффективным. Браузер будет загружать его один раз и затем помещать в кэш. Страницы будут просто указывать на этот находящийся в кэше экземпляр .css-файла и не будут больше содержать снова все стили, а это значит, что они будут меньше по размеру и, следовательно, будут быстрее загружаться. В некоторых случаях это может значительно ускорять загрузку Web-страниц в браузере пользователя.

Ниже приводится пример того, как можно переопределить стиль показанного выше элемента DIV путем сохранения его в отдельном файле с именем styles.css:

```
.mystyle
{
   align:            justify;
   color:            red;
   background-color: yellow;
   font-size:        12px;
}
```

Далее нужно связать CSS-файл с HTML в .aspx или .htm-странице, как показано ниже:

```
<head>
  <link href="/styles.css" text="text/css" rel="stylesheet" />
  <!-- другие метадескрипторы... -->
</head>
```

И, наконец, необходимо написать HTML-дескриптор div и указать, какой CSS-класс он должен использовать:

```
<div class="mystyle">некоторый текст</div>
```

Обратите внимание на то, что при объявлении стиля я использовал перед именем класса точку (.). Подобным образом следует поступать при создании всех специальных классов стилей.

Если нужно определить стиль, который должен применяться ко всем HTML-объектам определенного вида (например, ко всем параграфам <p> или даже ко всем дескрипторам <body> страницы), не имеющим другого явного ассоциируемого с ними класса, можно написать в файле таблицы стилей следующий код:

```
body
{
  margin:     0px;
  font-family: Verdana;
  font-size:  12px;
}

p
{
  align:      justify;
  text-size:  10px;
}
```

Этот код позволяет в одном месте установить стиль, который должен использоваться по умолчанию для всех дескрипторов <body> и <p>. Однако, вы запросто могли бы установить для некоторых дескрипторов <p> и какой-то другой стиль, просто указав в них имя явного класса.

Существует еще один способ связать класс стиля с HTML-объектом. Этот способ заключается в использовании идентификатора. Сначала определяется имя класса при помощи префикса #, как показано ниже:

```
#header
{
  padding:    0px;
  margin:     0px;
  width:      100%;
  height:     184px;
  background-image: url(images/HeaderSlice.gif);
}
```

Затем используется атрибут id HTML-дескриптора для связывания CSS с HTML. Например, вот как можно было бы определить HTML-дескриптор и указать, что он должен использовать стиль #header:

```
<div id="header">some text</div>
```

Обычно такой подход применяется для одиночных объектов, таких как верхний колонтитул, нижний колонтитул, контейнер для левой, правой, центральной колонки и т.п.

И, наконец, можно смешивать различные подходы. Предположим, что вы хотите установить какой-нибудь определенный стиль для всех ссылок в контейнере с классом sectiontitle и какие-то другие стили для всех ссылок в контейнере с классом sectionbody. Вы можете сделать это следующим образом:

В файле .css:

```
.sectiontitle a
{
   color: yellow;
}

.sectionbody a
{
   color: red;
}
```

В файле .aspx/.htm:

```
<div class="sectiontitle">
некоторый текст
<a href="http://www.wrox.com">Wrox</a>
некоторый текст

</div>
<div class="sectionbody">
другой текст
<a href="http://www.wiley.com">Wiley</a>
другой текст
</div>
```

Избегайте использования HTML-таблиц для управления схемой

Иногда разработчики для управления размещением элементов на Web-странице могут использовать HTML-таблицы. Такой подход считался стандартным до того, как появилась технология CSS, но многие разработчики до сих пор пользуются им. Хотя это и очень распространенный подход, W3C официально не одобряет его (`www.w3c.org/tr/wai-webcontent`), утверждая следующее: "Таблицы должны использоваться для размещения исключительно по-настоящему табличной информации (т.е. в качестве *таблиц данных*), и разработчики содержимого должны стараться избегать их применения для разметки страниц (т.е. в качестве *разметочных таблиц*). Кроме того, HTML-таблицы, используемые как в том, так и в другом качестве, представляют собой проблему для пользователей программ типа читателя экрана". Другими словами, HTML-таблицы следует применять для отображения на странице табличных данных, а не для создания всей схемы размещения элементов на странице. Поэтому лучше использовать элементы управления типа контейнера (такие как `DIV`) и их атрибут стиля, возможно, в отдельном разделе `<style>` или даже в отдельном файле. Такой подход является идеальным по нескольким причинам.

- Если вы используете контейнеры `DIV` и отдельный файл таблицы стилей для определения внешнего вида и месторасположения элементов на странице, вам не придется повторять это определение снова и снова для каждой страницы сайта. А это означает, что на разработку сайта уйдет гораздо меньше времени, да и обслуживать такой сайт будет гораздо удобнее.
- Сайт будет загружаться для конечных пользователей гораздо быстрее! Помните, что файл таблицы стилей будет загружаться клиентом только один раз и после этого, при последующих запросах страниц, он будет загружаться уже из кэша, и так до тех пор, пока в него не будут внесены изменения на сервере. Если вы определите схему размещения элементов внутри HTML-файла при помощи таб-

лиц, тогда клиент вместо этого будет загружать схему таблицы для каждой страницы и, следовательно, он будет загружать больше байтов, а это значит, что на загрузку страницы в целом будет уходить больше времени. Обычно управляемая CSS-файлом схема размещения элементов позволяет сократить количество загружаемых байтов на 50%, что сразу же делает очевидным преимущество такого подхода. Более того, подобная экономия может быть особенно выгодна на сильно загруженном сервере — умножьте количество байтов, отправляемых каждому пользователю, на количество подключающихся одновременно пользователей, и вы увидите, сколько байтов трафика вам может сэкономить CSS-подход на стороне Web-сервера.

- Программам типа читателя экрана (screen reader), которые могут читать текст и другое содержимое страницы для слепых или с ослабленным зрением пользователей, приходится гораздо сложнее, когда для размещения элементов на странице используются HTML-таблицы. Следовательно, применяя лишенную таблиц схему размещения, вы можете увеличить доступность сайта. Этот фактор является очень важным для сайтов определенных категорий, например таких, как сайты государственных ведомств и правительственных органов. Мало какие компании согласятся сбросить со счетов целые группы пользователей по таким пустяковым причинам, как эта.
- CSS-стили и контейнеры `DIV` обеспечивают большую степень гибкости, чем HTML-таблицы. Например, вы можете создать различные файлы таблиц стилей, по-разному определяющие внешний вид и месторасположение различных объектов на странице. Меняя подключенный файл таблицы стилей, вы сможете полностью изменять внешний вид страницы, совершенно ничего не изменяя на самих страницах содержимого. В случае динамических страниц ASP.NET вы сможете даже изменять файл таблицы стилей во время выполнения и, соответственно, легко реализовать механизм, позволяющий конечным пользователям выбирать те стили, которые им больше нравятся. И речь идет не только о цветах и шрифтах — вы также можете указывать в CSS-файлах позиции для объектов и, следовательно, иметь файл, размещающий поле меню в левом верхнем углу страницы, и еще один, размещающий его в правом нижнем углу страницы. Поскольку мы хотим позволить пользователям выбирать их любимые стили из списка доступных тем, этот пункт является для нас особенно важным.
- CSS позволяет в некоторых случаях охватывать различные классы устройств наподобие мобильных устройств типа карманных компьютеров и смартфонов, не создавая новой HTML-разметки. Из-за того, что такие устройства имеют ограниченный размер экрана, для них нужно подгонять вывод, так чтобы содержимое умещалось на маленьком экране, и чтобы его было удобно читать. Это можно сделать при помощи специальной таблицы стилей, изменяющей размер и местоположение некоторых контейнеров (путем размещения их одного под другим, а не в вертикальных колонках) или полностью скрывающей их. Например, вы можете скрыть контейнер для баннеров, опросов и верхнего колонтитула с большим логотипом. Попробуйте сделать это с HTML-таблицами — это будет гораздо сложнее. Вам придется подумать о специальном механизме скиннинга, а также написать отдельные страницы для всех схем размещения элементов, которые должны быть доступны. А на это понадобится гораздо больше усилий, нежели на то, чтобы просто написать новый CSS-файл.

Обратите внимание на то, что приведенная выше информация касается использования HTML-таблиц для общей схемы сайта. Однако применение таблиц для создания принимающих входные данные форм с табличной структурой вполне допускается, потому что CSS-кода для выполнения этой задачи так, чтобы формы легко перезаписывались и обслуживались, будет нужно слишком много. Также вероятность того, что у вас когда-нибудь возникнет необходимость динамически изменить схему формы, довольно низка, поэтому вся предоставляемая CSS гибкость вам для этого вовсе не нужна: использование HTML-таблиц будет в таком случае более удобным подходом.

Использование общего дизайна к множеству страниц

Закончив создавать красивый дизайн для своего сайта, нужно отыскать способ быстро применить его к n-ному количеству страниц, где под упомянутым количеством могут подразумеваться десятки или даже тысячи страниц. В предыдущем издании этой книги, посвященном ASP.NET 1.x, применялся классический подход, заключавшийся в выделении общих частей дизайна и помещении их в файлы пользовательских элементов управления для импортирования во все нуждающиеся в них страницы. В частности, у нас был пользовательский элемент управления для верхнего колонтитула и еще один — для нижнего колонтитула. Хотя этот подход был значительно лучше подхода, предусматривающего фактическую репликацию всего кода во всех страницах, и намного лучше подхода, подразумевающего включение файлов классической версии ASP (из-за их объектно-ориентированной природы), идеальным он все-таки не был. Проблема этого подхода состояла в том, что абсолютно для каждой страницы все равно нужно было писать в `.aspx`-файлах определенные строки для импорта элементов управления и для размещения этих элементов управления в нужном месте на странице. Соответственно, если элементы управления размещались в каком-нибудь одном месте на первой странице и в каком-нибудь другом на второй странице, во время выполнения эти две страницы выглядели по-разному. Обращать внимание на эти детали при создании каждой новой страницы содержимого очень утомительно; намного удобнее сконцентрировать внимание на содержимом для одной определенной страницы и затем просто применить общий дизайн ко всем остальным страницам автоматически. Т.е. необходим некий визуальный механизм наследования, позволяющий определять одну "базовую" страницу и делать так, чтобы остальные страницы просто наследовали ее схему. В ASP.NET 1.x механизм наследования мог применяться только на уровне отделенного кода и, следовательно, влиять только на поведение страницы (т.е. на то, какие операции должны выполняться, когда страница загружается, выгружается или визуализируется), но не на ее внешний вид. Существовали некоторые способы, позволявшие частично обходить эту проблему, но лично меня ни один из них устраивал ни в плане функциональных возможностей, ни в плане поддержки в период проектирования. В ASP.NET 2.0 эта проблема наконец-то была решена.

Модель мастер-страницы

ASP.NET 2.0 предоставляет новую функциональную возможность, называемую *мастер-страницей* (master page) и позволяющую определять общие области, которые будут присутствовать на каждой странице, наподобие верхних колонтитулов, нижних колонтитулов, меню и т.п. Мастер-страница позволяет помещать код общей схемы в один файл и заставлять его визуально наследоваться во все страницах содержимого. Мастер-страница содержит общую схему сайта. Страницы содержимого могут наследо-

вать внешний вид мастер-страницы и размещать свое собственное содержимое в тех местах, где на мастер-странице находится элемент управления ContentPlaceHolder. Хотя это создает впечатление предоставления некоего визуального механизма наследования, на самом деле такой механизм в понимании ООП (объектно-ориентированного программирования) не реализуется — вместо этого базовая реализация мастер-страниц основывается на модели шаблонов (template model).

Любой пример стоит тысячи слов, поэтому давайте посмотрим, как эта концепция выглядит на практике. Мастер-страница имеет расширение .master и в принципе похожа на пользовательский элемент управления. Ниже показан код мастер-страницы, которая содержит кое-какой текст, верхний колонтитул, нижний колонтитул и элемент управления ContentPlaceHolder между ними.

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="MasterPage.master.cs"
Inherits="MasterPage" %>

<html>
<head id="Head1" runat="server">
    <title>TheBeerHouse</title>
</head>

<body>
<form id="Main" runat="server">
   <div id="header">The Beer House</div>
   <asp:ContentPlaceHolder ID="MainContent" runat="server" />
   <div id="footer">Copyright 2005 Marco Bellinaso</div>
</form>
</body>
</html>
```

Как видите, эта страница очень похожа на стандартную страницу, за исключением того, что у нее в начале идет директива @Master вместо директивы @Page, а на месте, где у .aspx-страниц добавляется их собственное содержимое, у нее объявляется один или более элементов управления ContentPlaceHolder. Во время выполнения мастер-страница и страница содержимого будут объединяться, а это значит, что, поскольку мастер-страница определяет дескрипторы <html>, <head>, <body> и <forms>, страницам содержимого это делать уже не нужно. Страницы содержимого будут определять только информационное наполнение для элементов управления ContentPlaceHolder мастер-страницы. Ниже показан пример страницы содержимого.

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
AutoEventWireup="true"
CodeFile="MyPage.aspx.cs" Inherits="MyPage"
Title="The Beer House - My Page" %>

<asp:Content ID="MainContent" ContentPlaceHolderID="MainContent" Runat="Server">
   Здесь идет содержимое страницы MyPage...
</asp:Content
```

Первым главным моментом здесь является то, что в директиве @Page устанавливается значение для атрибута MasterPageFile; это значение представляет собой виртуальный путь к мастер-странице, которая должна использоваться. Содержимое помещается в элементы управления Content, значение ContentPlaceHolderID которых должно соответствовать идентификатору (ID) одного из элементов управления ContentPlaceHolder мастер-страницы. На странице содержимого нельзя размещать

ничего, кроме элементов управления `Content`, и все остальные элементы управления ASP, которые фактически определяют визуальные функциональные возможности, должны группироваться под ними. Еще одним важным моментом, на который следует обратить внимание, является присутствие в директиве `@Page` атрибута `Title`, который позволяет перекрывать значение, указанное в метадескрипторе `<title>` мастер-страницы. Если вы не зададите атрибут `Title` для какой-нибудь страницы содержимого, тогда вместо него будет использоваться заголовок, указанный на главной странице.

На рис. 2.2 функциональная возможность мастер-страницы показана в графическом виде.

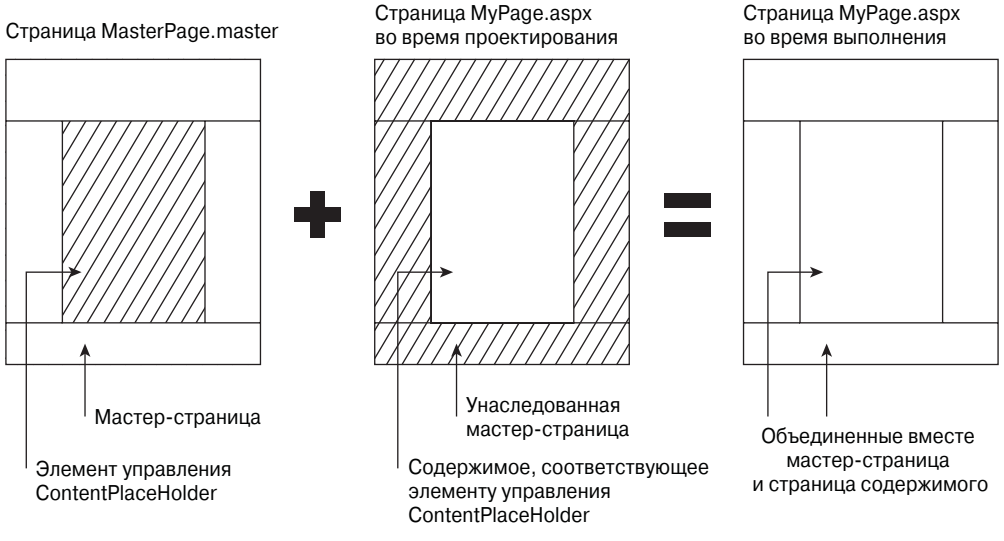

Рис. 2.2. Мастер-страница во время проектирования и во время выполнения

При редактировании страницы содержимого в Visual Studio в конструкторе форм визуализируются обе страницы, как мастер-страница, так и страница содержимого, но содержимое мастер-страницы "затеняется". Это делается для того, чтобы вы помнили, что изменять содержимое, предоставляемое мастер-страницей, во время редактирования страницы содержимого нельзя.

Еще я хотел бы обратить внимание на то, что у мастер-страницы также имеется файл code-behind; этот файл может использоваться для написания каких-нибудь C#-свойств или функций, к которым после этого можно будет получать доступ в файлах `.aspx` или code-behind страниц содержимого.

Определяя элемент управления `ContentPlaceHolder` в мастер-странице, также можно указать для него стандартное содержимое (т.е. содержимое, отображаемое по умолчанию), которое будет использоваться в случае отсутствия у той или иной страницы содержимого элемента управления `Content` для `ContentPlaceHolder`. Ниже показан пример того, как следует задавать стандартное содержимое.

```
<asp:ContentPlaceHolder ID="MainContent" runat="server">
  Здесь идет содержимое по умолчанию...
</asp:ContentPlaceHolder>
```

Стандартное содержимое может очень пригодиться в ситуации, когда требуется добавить новый раздел в ряд страниц содержимого, но изменить их все сразу возможности нет. В таком случае можно поступить следующим образом: создать в мастер-странице новый элемент управления ContentPlaceHolder, присвоить ему какое-нибудь содержимое по умолчанию и затем в спокойном темпе добавлять информацию в страницы содержимого — те страницы содержимого, которые еще не были изменены, будут просто отображать содержимое по умолчанию, предоставляемое мастер-страницей.

Атрибут MasterPageFile на уровне страницы тоже может быть полезен, если нужно сделать так, чтобы для разных наборов страниц содержимого использовались разные мастер-страницы. Однако если все страницы сайта должны использовать одну и ту же мастер-страницу, тогда установить ее для всех страниц будет проще из файла web.config, при помощи элемента <pages>, как показано ниже:

```
<pages masterPageFile="~/Template.master" />
```

Тем не менее, если все-таки указать атрибут MasterPageFile на уровне страницы, этот атрибут будет перекрывать значение, указанное в файле web.config для этой одной конкретной страницы.

Вложенные мастер-страницы

Можно пойти еще дальше и сделать так, чтобы мастер-страница выступала в роли содержимого для другой мастер-страницы. Другими словами, можно создавать вложенные мастер-страницы, где одна мастер-страница наследует внешний вид другой мастер-страницы, и где страницы содержимого .aspx наследуются от уже этой второй мастер-страницы. Мастер-страница второго уровня может выглядеть как-нибудь так:

```
<%@ Master Language="C#" MasterPageFile="~/MasterPage.master"
AutoEventWireup="true" CodeFile="MasterPage2.master.cs"
Inherits="MasterPage2" %>

<asp:Content ID="Content1" ContentPlaceHolderID="MainContent" Runat="Server">
   Какое-нибудь другое содержимое...
   <hr style="width: 100%;" />
   <asp:ContentPlaceHolder ID="MainContent" runat="server" />
</asp:Content>
```

Поскольку вы можете использовать один и тот же идентификатор для элемента управления ContentPlaceHolder на базовой мастер-странице и для еще одного элемента управления ContentPlaceHolder на унаследованной мастер-странице, изменять что-либо кроме значения атрибута MasterPageFile в странице содержимого для того, чтобы она использовала мастер-страницу второго уровня, вам не придется.

Эта возможность открывает большие перспективы, потому что позволяет создавать внешнюю мастер-страницу, определяющую самую общую структуру (обычно общекорпоративного уровня), и затем другие мастер-страницы, определяющие структуру определенных частей сайта, таких как раздел электронного магазина, раздел администрации и т.д. Единственной проблемой вложенных мастер-страниц является то, что они (в отличие от мастер-страниц первого уровня) не поддерживаются во время проектирования в IDE-среде Visual Studio. Редактируя страницы содержимого,

нужно пользоваться редактором в режиме Source View (Представление исходного кода), и увидеть результаты можно только в окне браузера при просмотре страницы. Для таких разработчиков как я, которые предпочитают писать большую часть кода самостоятельно в Source View, это не такая уж большая проблема, и иметь возможность использовать вложенные мастер-страницы — просто отличная вещь!

Получение доступа к мастер-странице из страницы содержимого

Также существует возможность получать доступ к мастер-странице из страницы содержимого, через такое свойство страницы, как `Master`. Это свойство возвращает объект типа `MasterPage`, который наследуется прямо от `UserControl` (помните, я говорил, что мастер-страницы похожи на пользовательские элементы управления) и включает несколько дополнительных свойств. Он предоставляет коллекцию `Controls` (элементы управления), которая позволяет получать доступ к элементам управления мастер-страницы из страницы содержимого. Это может быть необходимо, например, в случае, если нужно программно скрыть какие-нибудь элементы управления мастер-страницы (вроде поля для регистрации пользователя или поля для баннера) на какой-нибудь конкретной странице. Получение доступа к коллекции `Controls` напрямую будет возможным, но потребует ручного приведения возвращаемого объекта `Control` к соответствующему типу элемента управления и, следовательно, будет представлять собой слабо типизированный подход. Более удачным и объектно-ориентированным подходом будет добавить в класс code-behind мастер-страницы специальные свойства — в нашем примере это будет свойство `Visible`. Вот что вы могли бы написать:

```
public bool LoginBoxIsVisible
{
   get { return LoginBox.Visible; }
   set { LoginBox.Visible = value; }
}
```

Теперь, в странице содержимого, вы можете добавить после директивы `@Page` следующую строку:

```
<%@ MasterType VirtualPath="~/MasterPage.master" %>
```

С помощью этой строки вы указываете путь к мастер-странице, используемой ASP.NET во время выполнения для динамической генерации строго типизированного класса `MasterPage`, который отображает специальные свойства, добавленные в его класс code-behind. Я знаю, что это похоже на копию атрибута `MasterPageFile` директивы `@Page`, но только так вы можете сделать свойства мастер-страницы видимыми на странице содержимого. Указать тип мастер-страницы можно не только при помощи виртуального пути (как в показанном выше примере), но и при помощи имени класса мастер-страницы, посредством атрибута `TypeName`. Добавив эту директиву в файл code-behind страницы содержимого (или в раздел `<script runat="server">` самого файла `.aspx`), вы сможете легко получить доступ к свойству `LoginBoxIsVisible` мастер-страницы в строго типизированной манере, как показано ниже:

```
protected void Test_OnClick(object sender, EventArgs e)
{
   this.Master.LoginBoxIsVisible = false;
}
```

Когда я говорю "в строго типизированной манере", я имею в виду, что на этом свойстве у вас сработает Visual Studio Intellisense: введите "this.Master." и, когда вы введете вторую точку, вы увидите ваше новое свойство в списке Intellisense.

Такой способ получения доступа к объектам мастер-страницы из страниц содержимого также особенно удобен, когда требуется включить в мастер-страницу какие-нибудь общие методы для того, чтобы они использовались всеми ассоциируемыми с ней страницами. Если бы у нас не было доступа к строго типизированному объекту `MasterPage`, создаваемому ASP.NET во время выполнения, для получения доступа к этим методам нам пришлось бы применять технологию рефлексии, которая является более медленной и, конечно же, намного менее удобной в использовании (например, в данном случае, было бы гораздо проще поместить совместно используемые методы в отдельный класс, к которому могла бы получать доступ каждая страница).

Тем из вас, кто читал первое издание этой книги, я хочу указать на разницу между использованием объектно-ориентированной базовой страницы (OOP Page) и использованием мастер-страницы (Master Page). В первом издании мы определяли базовый класс под названием `ThePhile`, который наследовался всеми страницами типа "содержимое". Это было самое настоящее объектно-ориентированное наследование в действии, но область применения этого класса была ограниченной, потому что мы не могли наследовать от него никакие элементы, касающиеся внешнего вида. Мы все равно были вынуждены создавать пользовательские элементы управления для получения общих визуальных элементов. Однако в ASP.NET 2.0, когда мы определяем мастер-страницу, мы можем добиться наследования всех визуальных элементов (но не наследования объектно-ориентированного кода). Отсутствие механизма наследования кода не является серьезным ограничением, поскольку у нас есть возможность получать доступ к этому коду в мастер-странице через ссылку `MasterType`, как объяснялось выше.

Переключение мастер-страниц во время выполнения

Последнее, о чем я хочу рассказать в этом посвященном знакомству с мастер-страницами разделе, это о возможности динамически менять используемую страницей содержимого мастер-страницу во время выполнения. Все правильно, вы можете иметь множество мастер-страниц и выбирать, какая из них должна использоваться, уже после запуска сайта. Это делается путем установки свойства `MasterPageFile` страницы из ее обработчика события `PreInit`, как показано ниже.

```
protected void Page_PreInit(object sender, EventArgs e)
{
   this.MasterPageFile = "~/OtherMasterPage.master";
}
```

Событие `PreInit` является новым в ASP.NET 2.0, и устанавливать свойство `MasterPageFile` можно только в этом обработчике события, потому что объединение двух страниц должно происходить как можно раньше в жизненном цикле страницы (на стадии события `Load` или `Init` будет слишком поздно).

Меняя мастер-страницу динамически, нужно обязательно убедиться в том, что все мастер-страницы имеют одинаковый идентификатор (ID) для элементов управления `ContentPlaceHolder`, таким образом, элементы управления `Content` страницы содержимого всегда будут соответствовать им, независимо от того, какая из мастер-страниц используется. Этот замечательный подход дает возможность создавать множество мастер-страниц, задающих совершенно разные схемы, и тем самым давать возмож-

ность пользователям выбирать ту из них, которая им нравится больше. Недостатком этого подхода является то, что если написать специальный код в файле code-behind мастер-страницы, его придется реплицировать в классе code-behind всех остальных страниц; иначе страница содержимого будет не всегда отыскивать его. Кроме того, нельзя будет использовать строго типизированное свойство `Master`, потому что изменять тип мастер-страницы динамически во время выполнения не разрешено; его можно только устанавливать при помощи директивы `@MasterType`. По этим причинам мы не будем использовать разные мастер-страницы для предоставления разных схем пользователю. Вместо этого, у нас будет только одна страница, к которой мы сможем применять разные файлы таблиц стилей. Поскольку мы решили использовать лишенную таблиц схему, мы сможем полностью изменять внешний вид страницы (шрифты, цвета, изображения и позиции) путем применения к ней различных стилей.

Создание набора выбираемых пользователем тем

Темы — это новая функциональная возможность в ASP.NET 2.0, которая позволяет пользователям иметь еще больший контроль над внешним видом и поведением Web-страницы. Тема может применяться для определения цветовых схем, названий шрифтов, размеров и стилей и даже изображений (квадратные или скругленные углы либо изображения с разными цветами и оттенками). Новая поддержка Skin (Обложка) в ASP.NET 2.0 — это расширение идеи, лежащей в основе CSS. Пользователи могут выбирать тему из различных доступных им опций, а тема, которую они выбирают, определяет "обложку", которая указывает, какие визуальные стилистические параметры будут использоваться для их сеанса пользователя. Обложки являются серверным аналогом CSS-таблиц. Файл обложки похож на CSS-файл, но, в отличие от CSS, обложка может перекрывать различные визуальные свойства, которые были явно установлены на серверных элементах управления внутри страницы (глобальная CSS-спецификация никогда не сможет перекрыть набор стилей, установленный на определенном элементе управления). Вы можете сохранять с темами специальные версии изображений, что может оказаться удобным, когда нужны несколько наборов изображений, использующих разную цветовую схему на основании текущей обложки. Однако темы не исключают необходимость в использовании CSS-файла; вы можете использовать и CSS-файлы, и файлы обложек вместе для достижения максимальной гибкости и степени контроля. Что касается файлов таблиц стилей, в версии ASP.NET 2.0 для них почти ничего не изменилось, кроме того, что появилось еще несколько элементов управления, позволяющих указывать свойство `CssClass`, и того, что теперь еще несколько элементов управления имеют поддержку визуального конструктора, т.е. позволяют выбирать "заготовленную" CSS-спецификацию.

Тема — это группа связанных между собой файлов, хранящихся в отдельной подпапке внутри папки сайта, которая называется `/App_Themes` и может содержать следующие элементы.

- Файлы таблиц стилей (`.css`), отвечающие за внешний вид объектов HTML.
- Файлы обложек, отвечающие за внешний вид серверных элементов управления ASP.NET. Можете считать их серверными файлами таблиц стилей.
- Другие ресурсы, такие как изображения.

Одним из преимуществ способа, которым ASP.NET 2.0 реализует темы, является то, что, когда вы применяете к странице тему (о том, как это делается, я расскажу

чуть ниже), ASP.NET автоматически создает на каждой странице метадескриптор <link> для каждого .css-файла, находящегося в папке данной темы, во время выполнения! Это хорошо, потому что вы можете переименовать тот или иной существующий CSS-файл или добавить новый, и все ваши страницы все равно будут автоматически указывать друг на друга. Этот момент имеет особое значение, потому что, как будет показано, вы можете динамически изменять тему во время выполнения (как вы это можете делать и с мастер-страницей), а ASP.NET будет связывать файлы в папке новой темы, тем самым изменяя внешний вид сайта в соответствии с предпочтениями отдельных пользователей. Без этого механизма вы должны были бы вручную создавать все метадескрипторы <link> во время выполнения в соответствии с выбираемой пользователем темой, что потребовало бы немалых усилий.

Самой лучшей функциональной возможностью в категории тем являются новые серверные таблицы стилей, называемые *файлами обложек*. Эти файлы имеют расширение .skin и содержат объявление элементов управления ASP.NET, как показано ниже:

```
<asp:TextBox runat="server" BorderStyle="Dashed" BorderWidth="1px" />
```

Объявление, помещаемое в файл обложки, ничем не отличается от обычного объявления, помещаемого в файл .aspx, за исключением того, что в нем не указываются идентификаторы элементов управления. Как только к странице (или страницам) применяется тема, ее (их) элементы управления приобретают внешний вид определений, написанных в файле (файлах) обложки. В случае элемента управления TextBox эта возможность может и не показаться такой уж впечатляющей, потому что подобного эффекта можно добиться и просто написав класс стиля для соответствующего HTML-элемента <input> в .css-файле. Однако, как только вы осознаете, что вы можете делать это для более сложных элементов управления, таких как Calendar или DataGrid (а также нового элемента управления GridView), вы увидите, что такая возможность является гораздо более важной, поскольку такие элементы управления не могут состоять в отношениях типа "один к одному" с HTML-элементом и, соответственно, определить их стиль просто путем написания одного единственного класса в классическом .css-файле нельзя.

Вы можете создать как один единственный файл .skin и поместить в него определение для элементов управления всех типов, так и множество файлов .skin, по одному для каждого типа элементов управления, например таких: TextBox.skin, DataGrid.skin, Calendar.skin и т.п. Во время выполнения все эти файлы все равно будут объединятся вместе в памяти, так что это просто дело вкуса: поступайте так, как вам больше нравится.

Применить тему к одной единственной странице можно, используя атрибут Theme в директиве @Page:

```
<%@ Page Language="C#" Theme="NiceTheme"
MasterPageFile="~/MasterPage.master" ... %>
```

Применить тему ко всем страницам можно, установив атрибут theme элемента <pages> в файле web.config, как показано ниже:

```
<pages theme="NiceTheme" masterPageFile="~/MasterPage.master" />
```

Что касается мастер-страниц, для них тему также можно изменять и программно, из события PreInit класса Page. Например, ниже показано, как применяется тема, имя которой содержится в переменной Session.

```
protected void Page_PreInit(object sender, EventArgs e)
{
   if (this.Session["CurrentTheme"] != null)
      this.Theme = this.Session["CurrentTheme"];
}
```

В главе 4 мы улучшим этот механизм, заменив переменные Session на новые свойства Profile.

Когда используется атрибут Theme директивы @Page (или атрибут theme в файле web.config), атрибуты внешнего вида, указываемые в файле (файлах) обложки перекрывают аналогичные атрибуты, заданные в файлах .aspx. Если вы хотите, чтобы темы работали как таблицы стилей .css (т.е., чтобы вы могли определять стили в файлах .skin и перекрывать их в файлах .aspx для определенных элементов управления), вы можете добиться такого эффекта, указывая тему при помощи атрибута StylesheetTheme директивы @Page или при помощи атрибута styleSheetTheme элемента <page> в файле web.config. Старайтесь не путать атрибут Theme с атрибутом StylesheetTheme.

До сих пор я описывал *безымянные* обложки, т.е. обложки, которые определяют внешний вид всех элементов управления какого-нибудь определенного типа. Однако в некоторых случаях вам будет нужно, чтобы внешний вид какого-нибудь элемента управления отличался от того, который вы указали для элементов управления такого типа в файле обложки. Существуют три способа, которыми вы можете добиться такого эффекта.

1. Как уже упоминалось ранее, вы можете применить тему при помощи свойства StylesheetTheme (вместо свойства Theme) для того, чтобы визуальные свойства, указываемые в .aspx-файлах, перекрывали те, что заданы в файле обложки. Однако по умолчанию механизм тем обеспечивает одинаковый внешний вид для всех элементов управления каждого типа; это поведение создавалось с расчетом на ситуации, когда разработкой страниц занимается несколько человек, и быть уверенным в том, что все они используют атрибуты в файлах .aspx нельзя, кроме тех случаев, когда их применение является обязательным.
2. Отключить механизм тем конкретно для данного элемента управления и применить атрибуты внешнего вида как обычные, как показано в следующем коде:

```
<asp:TextBox runat="server" ID="btnSubmit" EnableTheming="False"
   BorderStyle="Dotted" BorderWidth="2px" />
```

3. Использовать для этого элемента управления именованную обложку, которая представляет собой определение обложки с атрибутом SkinID, как показано ниже:

```
<asp:Label runat="server" SkinID="FeedbackOK" ForeColor="green" />
<asp:Label runat="server" SkinID="FeedbackKO" ForeColor="red" />
```

Объявляя элемент управления, вы должны будете использовать соответствующее значение для его свойства SkinID, как показано ниже:

```
<asp:Label runat="server" ID="lblFeedbackOK"
   Text="Your message has been successfully sent."
   <!-- Text="Ваше сообщение успешно отправлено." -->
SkinID="FeedbackOK" Visible="false" />

<asp:Label runat="server" ID="lblFeedbackKO"
   Text="Sorry, there was a problem sending your message."
   <!-- Text="При отправке вашего сообщения возникла проблема." -->
SkinID="FeedbackKO" Visible="false" />
```

На мой взгляд, этот способ является самым лучшим, потому позволяет определять для одного и того же типа элементов управления множество стилей (внешних видов), в одном единственном файле, и затем применять их к любой странице. Кроме того, если вы будете хранить все определения стилей в файлах обложек, а не в файлах самих страниц, вы сможете полностью изменять внешний вид и поведение сайта, просто меняя текущую тему (что и является главной задачей тем). С жестко закодированными стилями, подобное возможно только частично.

В разделе "Решение" этой главы темы будут использоваться для создания нескольких разных визуальных представлений для одной и той же мастер-страницы.

Создание навигационной системы

Как говорилось в разделе "Задача", вам необходимо отыскать какой-нибудь способ создать систему меню, удобную в обслуживании и понятную для пользователей. Вы, наверное, думаете, что можете просто жестко закодировать меню как HTML-код, но это далеко не наилучший вариант, потому что это означает, что вам придется копировать и вставлять код, если вы захотите, чтобы меню отображалось в более чем каком-то одном месте (в данном случае, в верхнем и в нижнем колонтитулах), а также если вы захотите добавить или изменить какую-нибудь ссылку. В первом издании этой книги мы создавали специальный элемент управления, который брал XML-файл, содержащий карту сайта (т.е. имя и URL-адрес ссылок, отображаемых в меню), и генерировал HTML-код, который должен визуализироваться на странице, путем применения к нему файла XSL. ASP.NET 2.0 предлагает кое-какие новые встроенные элементы управления и функциональные средства, которые позволяют делать примерно то же самое, но с большим количеством возможностей, что значительно упрощает жизнь разработчику.

Определение файла карты сайта

Опции меню указываются в XML-файле карты сайта. Главный файл карты для всего сайта называется web.sitemap и представляет собой иерархическую структуру узлов <siteMapPath>, которые имеют атрибуты title и url. Ниже показан пример такого файла.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
   <siteMapNode title="Home" url="~/Default.aspx" />
   <siteMapNode title="About" url="~/About.aspx" />
   <siteMapNode title="Contact" url="~/Contact.aspx" />
</siteMap>
```

В нашем случае будут предусмотрены узлы первого уровня типа Home (На главную), Contacts (Контакты) и About (О нас), а также узлы второго и, возможно, третьего уровня типа Store/Shopping Cart (Корзина для покупок). Ниже показан более сложный пример файла web.sitemap с записями второго уровня и ссылкой на еще один дочерний файл sitemap, который предоставляет опции меню для Store.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode title="Books" url="~/Books/Books.aspx>
    <siteMapNode title="Authors" url="~/Books/Authors.aspx" />
    <siteMapNode title="Publishers" url="~/Books/Publishers.aspx" />
  </siteMapNode>
  <siteMapNode title="DVDs" url="~/DVDs/DVDs.aspx>
    <siteMapNode title="Movies" url="~/DVDs/Movies.aspx" />
    <siteMapNode title="Songs" url="~/DVDs/Songs.aspx" />
  </siteMapNode>
  <siteMapNode siteMapFile="~/StoreMenu.sitemap" />
</siteMap>
```

Дочерний файл siteMap (StoreMenu.sitemap) имеет точно такой же формат, как и файл web.sitemap , а также внешний узел siteMap.

Привязывание файла sitemap к элементам управления типа меню

Определив файл sitemap, вы можете использовать его в качестве источника данных для новых элементов управления ASP.NET 2.0, таких как Menu и TreeView. ASP.NET 2.0 также предлагает новые не визуальные элементы управления, которые называются элементами управления DataSource и позволяют устанавливать связь с базой данных, XML-файлом и классом компонентов. Эти элементы управления будут использоваться графическими элементами управления для извлечения данных, которые должны привязываться и отображаться на экране. На практике они служат "мостиком" между хранилищем фактических данных и визуальным элементом управления. Одним из таких элементов управления DataSource является элемент управления SiteMapDataSource, который предназначен специально для файла сайта web.sitemap и определяется следующим образом:

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" runat="server" />
```

Обратите внимание на то, что путь к файлу web.sitemap не указывается, потому что для всего сайта может существовать только одна единственная карта, название которой выглядит как ~/web.sitemap. Когда создаются дочерние карты сайта для определенных подпапок внутри сайта, привязка выполняется автоматически, потому что связь с ними устанавливается из файла web.sitemap.

Если вам не нравится, как работает стандартный элемент управления SiteMapDataSource (например, потому, что вы хотите, чтобы поддерживалось множество файлов sitemap, или потому, что вы хотите, чтобы карта сайта хранилась не в XML-файле, а в базе данных), вам придется написать новый элемент DataSource или создать новый класс поставщика, явно предоставляющий содержимое для элемента управления SiteMapDataSource.

Элемент управления Menu позволяет создавать популярные всплывающие DHTML-меню с вертикальной или горизонтальной ориентацией. В ASP.NET 1.1 приличных стандартных элементов управления Menu не было, и поэтому все компании, занимающиеся разработкой Web-компонентов, предлагали свои собственные компоненты для

создания таких меню. Однако в ASP.NET 2.0 имеется стандартный элемент управления Menu, доступный бесплатно и способный интегрироваться с различными элементами управления типа DataSource. Например, создать элемент управления Menu, привязанный к определенному выше элементу управления SiteMapDataSource, можно при помощи такой простой строки:

```
<asp:Menu ID="mnuHeader" runat="server" DataSourceID="SiteMapDataSource1" />
```

Конечно, элемент управления Menu имеет много свойств, которые позволяют указывать его ориентацию (свойство Orientation), класс CSS, который он должен использовать (свойство CssClass), или стили для его различных частей, количество внутренних уровней, которые будут отображаться при его развертывании (свойство StaticDisplayLevels), и многое другое. Однако подробное рассмотрение всех этих свойств выходит за рамки контекста данной книги; поэтому всю необходимую дополнительную информацию следует искать в официальной документации MSDN.

Можно сделать так, чтобы карта сайта отображалась не в виде разворачивающихся меню, а в виде древовидной структуры; для этого нужно просто заменить объявление элемента управления Menu на новый элемент управления TreeView, как показано ниже:

```
<asp:TreeView runat="server" ID="tvwMenu" DataSourceID="SiteMapDataSource1" />
```

Панель "хлебные крошки"

Помимо меню, вы также можете захотеть, чтобы у пользователей была какая-нибудь визуальная подсказка по поводу того, где они сейчас находятся, а также какой-нибудь способ, позволяющий им вернуться с того места, где они находятся, обратно на главную страницу. Обычно такого эффекта можно добиться путем использования *хлебных крошек*, т.е. панели навигации, отображающей ссылки на все страницы (начиная с главной) или разделы, которые посещал пользователь, прежде чем попасть на текущую страницу, как показано ниже:

```
Home > Store > Shopping cart
```

При наличии такой навигационной системы, пользователь сможет возвращаться на две страницы обратно, не нажимая кнопку Back (Назад) в панели браузера (которая может быть и не видна по ряду причин), а также не возвращаясь снова на самую первую, главную страницу и не пытаясь запомнить путь, который он прошел ранее. В версии ASP.NET 2.0 панель “хлебные крошки” можно добавить при помощи одной единственной строки кода, просто объявив на странице экземпляр нового элемента управления SiteMapPath:

```
<asp:SiteMapPath ID="SiteMapPath1" runat="server" />
```

Как обычно, этот элемент управления имеет ряд свойств, которые позволяют полностью настраивать его внешний вид и поведение, как вы увидите на практике в разделе “Решение”.

Создание доступных сайтов

Все встроенные стандартные элементы управления в ASP.NET 2.0 по умолчанию визуализируют хорошо отформатированный код XHTML 1.0 Transitional. XHTML-код — это, по сути, HTML-код, написанный как XML, и как таковой он должен сле-

довать более строгим синтаксическим правилам. Например, все значения атрибутов должны заключаться в двойные кавычки, все дескрипторы должны иметь закрывающий дескриптор или являться явно самозакрывающимися (т.е., выглядеть не так:
 и <img src="...">, а так:
 и), а вложенные дескрипторы должны закрываться в правильном порядке (т.е. не так: hello Marco , а так: Hello Marco). Кроме того, многие HTML-дескрипторы, предназначенные для форматирования текста, такие как <font>, , <s> и т.д., теперь считаются устаревшими и должны заменяться CSS-стилями (типа font-family: Verdana; text-align: center). То же касается и некоторых атрибутов других дескрипторов, таких как width и align, например. Причина ввода такого нового стандарта проста: добиться большего разграничения между представлением и содержимым (о котором я уже говорил ранее) и создать более "чистый" код — код, который смогут читать программы, работающие с XML-данными. Тот факт, что ASP.NET 2.0 автоматически визуализирует XHTML-код, когда используются ее элементы управления, означает для разработчика возможность сэкономить массу времени и делает процесс привыкания к XHTML менее болезненным. Официальную документацию W3C о XHTML 1.0 можно найти по адресу: http://www.w3.org/TR/xhtml1/.

Что касается доступности, W3C называет ряд правил, призванных облегчить использование сайта для пользователей с физическими недостатками. Официальная страница руководства Web Content Accessibility Guidelines 1.0 (Руководство по обеспечению удобства использования Web-содержимого, версия 1.0), также часто называемого просто WCAG, находится по адресу www.w3.org/TR/WCAG10/. Рекомендации, известные как Section 508 (Раздел 508) были взяты именно из этого руководства и обязательно должны соблюдаться сайтами федеральных агентств США. Узнать больше об этом можно по адресу www.section508.gov/. Например, разработчики обязательно должны использовать в дескрипторах <img> атрибут alt для предоставления альтернативного текста пользователям с плохим зрением, т.е. для того, чтобы программы типа Screen Reader могли описывать изображения таким пользователям, а также они обязательно должны использовать дескриптор <label> для ассоциирования с полем для ввода соответствующей надписи. Другие рекомендации являются более трудновыполнимыми и не имеют непосредственного отношения к ASP.NET, поэтому тем, кто хочет узнать больше о них, следует воспользоваться официальной документацией. ASP.NET 2.0 делает вполне возможным выполнение некоторых простых правил, таких как те, о которых упоминалось выше. Например, у элемента управления Image появилось новое свойство GenerateEmptyAlternateText, которое, когда для него устанавливается значение true, генерирует alt="" (параметр AlternateText="" ничего бы не генерировал), а у элемента управления Label появилось новое свойство AssociateControlID, для которого в качестве значения устанавливается имя элемента управления <input> и которое во время выполнения генерирует для него элемент управления <label> (это свойство следует использовать вместе со свойством AccessKey, для создания клавиатурных команд для принимающего входные данные поля).

Если вы хотите узнать больше об XHTML, доступности и новых функциональных возможностях ASP.NET 2.0, имеющих отношение к этой теме, вы можете прочитать следующие доступные бесплатно онлайновые статьи: *Accessibility Improvements in ASP.NET 2.0 – Part 1* (ASP.NET 2.0: Улучшения в плане доступности. Часть 1) (www.15seconds.com/issue/040727.htm) и *Accessibility Improvements in ASP.NET 2.0 – Part 2* (ASP.NET 2.0: Улучшения в плане доступности. Часть 2) (www.15seconds.com/

issue/040804.htm) Алекса Хомера (Alex Homer), а также *Building ASP.NET 2.0 Web Sites Using Web Standards* (Создание Web-сайтов в ASP.NET 2.0 с использованием Web-стандартов) (http://msdn.microsoft.com/asp.net/default.aspx?pull=/library/en-us/dnaspp/html/aspnetusstan.asp) Стивена Уолтера (Stephen Walther).

Использование общего поведения ко всем страницам

Мастер-страницы и темы предоставляют поистине замечательную возможность, позволяя делать так, чтобы все страницы сайта имели одинаковый дизайн и внешний вид. Однако также может понадобиться, чтобы страницы имели какое-нибудь одинаковое поведение, т.е. чтобы на определенном этапе своего жизненного цикла они выполняли какой-то одинаковый код. Например, если вы захотите фиксировать доступ, получаемый ко всем страницам, для того, чтобы иметь возможность создавать и показывать статистику для своего сайта, вам придется выполнить соответствующий код при загрузке страницы. Еще одним случаем, когда вам может быть необходимо, чтобы для каждой страницы выполнялся какой-то определенный код, является случай, когда вы хотите установить свойство Theme страницы в обработчике событий PreInit. Да, вы действительно можете изолировать общий код во внешней функции и просто добавить соответствующую строку кода, которая будет вызывать ее (т.е. эту функцию с содержащимся в ней общим кодом) из каждой страницы, но такой подход имеет два недостатка.

- Вы не должны никогда забывать вставить эту вызывающую внешнюю функцию строку, когда создаете новую страницу. Если в создании .aspx-страниц принимает участие множество разработчиков (как чаще всего и бывает), вам придется проверять, чтобы никто из них не забыл об этой строке.
- Вы можете захотеть, чтобы из события PreInit выполнялся какой-нибудь один код инициализации, а из события Load — другой. В таком случае вам придется написать два отдельных метода xxxInitialize и добавить в каждую страницу еще две строки для вызова соответствующего метода из соответствующего обработчика событий. Потому не рассчитывайте особо на то, что добавить одну единственную строку в каждую страницу будет легко, потому что позже строк, которые нужно добавить, может оказаться гораздо больше. А когда количество страниц исчисляется сотнями (и я уверен, что вы со мной согласитесь), возврат и изменение всех страниц для добавления этих строк становится практически неосуществимой задачей.

Этих двух недостатков вполне достаточно для того, чтобы отказаться от такого подхода. Существует еще один подход, который заключается в написании общего кода в файле code-behind мастер-страницы. Такой подход может быть очень хорошим вариантом во многих ситуациях. Однако никак не в нашем случае, потому что мы должны обработать событие PreInit, а у класса MasterPage (и его базовых классов) такого события нет. Мы можем обработать событие Init или Load, но не событие PreInit, поэтому нам нужно придумать что-нибудь еще.

В предыдущем издании этой книги мы использовали класс BasePage, сделав так, чтобы все страницы содержимого наследовались от него, а не от стандартного класса System.Web.UI.Page. Я по-прежнему считаю, что такой подход является наилучшим, потому что из этого класса можно обрабатывать любые события, просто перекрывая методы OnXXX, где XXX — это имя события.

Ниже показана основная схема такого базового класса, который наследуется от класса `Page` и перекрывает методы `OnPreInit` и `OnLoad`.

```
public class BasePage : System.Web.UI.Page
{
   protected override void OnPreInit(EventArgs e)
   {
      // здесь нужно добавить специальный код...
      base.OnPreInit(e);
   }
protected override void OnLoad(EventArgs e)
   {
      // здесь нужно добавить специальный код...
      base.OnLoad(e);
   }
}
```

Далее классы в файлах code-behind страниц будут наследоваться от этого специального класса `BasePage`, а не от стандартного класса `Page`, как показано ниже:

```
public partial class Contact : BasePage
{
   // здесь нужно добавить обычный код страницы...
}
```

Внести какие-то изменения в файл code-behind класса каждой страницы все равно придется, но, сделав это, вы сможете позже возвращаться к классу `BasePage`, добавлять код в существующие методы или перекрывать новые методы и не изменять больше никаких дополнительных строк в файлах code-behind. Если вы выберете этот подход изначально, вы будете изменять классы code-behind один за другим по мере их создания, что будет легко и сделает ваш проект перспективным, т.е. гибким и удобным в плане внесения изменений.

Решение

К этому моменту вы уже должны иметь четкое представление о том, что вам нужно создать и как это делается, поэтому давайте приступим к разработке решения! Ранее в этой главе я объяснял, как вы можете создать предварительную модель своего сайта при помощи графического приложения типа Photoshop или Paint Shop Pro и сохранить ее в PSD-файле. Получив одобрение и разрешение продолжать, вы должны поделить изображения из PSD-файла на файлы `.gif` и `.jpg`, на которые можно напрямую ссылаться в файле Web-страницы. Какой бы метод вы не применяли для создания своих изображений, далее вы можете использовать их для создания Web-сайта. Первым делом вы должны создать новый проект Web-сайта, затем — мастер-страницу, главную страницу и тему, которая будет использоваться по умолчанию. Дальше вы можете создать вторую тему для сайта и реализовать механизм для изменения тем во время выполнения.

Итак, создайте сначала в Visual Studio .NET 2005 новый проект Web-сайта (чтобы сделать это, откройте меню File (Файл) и выберите в нем пункт New⇨Web Site⇨ ASP.NET Web Site (Создать⇨Web-сайт⇨Web-сайт ASP.NET)). А вот еще одна новая функциональная возможность в Visual Studio 2005: вы можете создать проект, указав папку в

файловой системе (вместо Web-адреса), если выберете в выпадающем списке Location (Размещение) значение File System (Файловая система), как показано на рис. 2.3.

Эта возможность позволяет создавать проект ASP.NET, не создавая соответствующего виртуального приложения или виртуального каталога в метабазе IIS-сервера (метабаза — это место, где IIS-сервер хранит свои конфигурационные данные); проект загружается из реальной папки на жестком диске и выполняется упрощенным интегрированным Web-сервером (под названием ASP.NET Development Server (Сервер разработки ASP.NET)), обрабатывающим запросы через TCP/IP-порт, отличный от того, которым пользуется IIS-сервер (IIS-сервер работает с портом 80). Номер используемого порта определяется произвольным образом каждый раз, когда вы нажимаете клавишу <F5>, чтобы запустить Web-сайт в режиме отладки. Например, этот сервер обрабатывает запросы типа `http://localhost:1168/ProjName/Default.aspx`. Это намного упрощает операции перемещения и резервного копирования проектов, потому что вы можете просто скопировать папку проекта и все — выполнять какие-то операции из консоли управления IIS-сервера (IIS Management) нет никакой необходимости. На самом деле Visual Studio 2005 даже не требует наличия IIS-сервера, если только вы не планируете развертывать свой сайт на Web-сервере IIS или не указываете вместо локального пути URL-адрес, когда создаете проект Web-сайта.

Если вам доводилось иметь дело с любой из предыдущих версий ASP.NET или Visual Studio 2005 (VS2005), я уверен, что вам понравится эта опция. Я назвал эту функциональную возможность опцией потому, что проект по-прежнему можно создавать путем использования в качестве пути к проекту URL-адреса (т.е. создавать и выполнять сайт под управлением IIS), выбирая в выпадающем списке Location (Размещение) значение HTTP (HTTP-протокол). Я рекомендую вам создавать и разрабатывать сайт, используя значение File System (Файловая система) и интегрированный Web-сервер, а потом уже, для стадии тестирования, переходить на полнофункциональный Web-сервер IIS. VS2005 включает новый мастер развертывания, который упрощает процедуру развертывания готового решения как на локальном, так и на удаленном Web-сервере IIS. Однако пока что просто создайте новый проект Web-сайта ASP.NET в какой хотите папке и назовите его TheBeerHouse (Пивная).

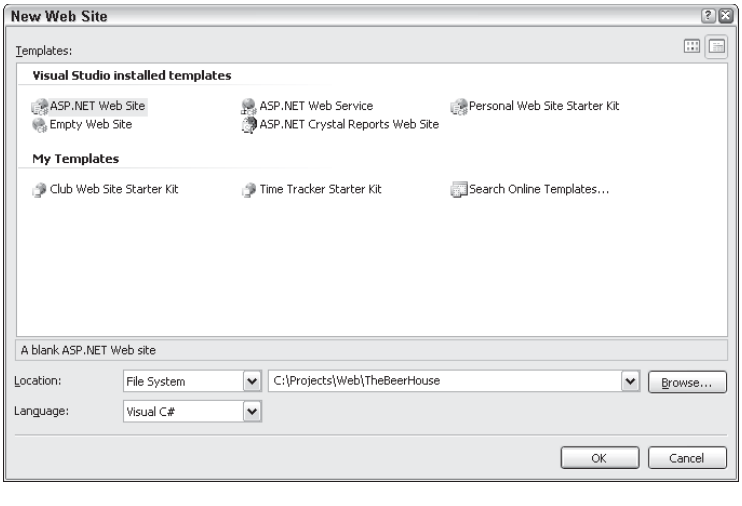

Рис. 2.3. Создание нового проекта Web-сайта

Интегрированный Web-сервер был разработан для упрощения процесса разработки и первоначального тестирования. Однако использовать его для окончательного тестирования типа тестирования на соответствие техническим требованиям (Quality Assurance Test) или на совместимость компонентов (Integration Test) нельзя. Для этого следует использовать IIS-сервер. IIS-сервер имеет больше возможностей вроде кэширования, HTTP-сжатия и многочисленных опций безопасности, из-за которых ваш сайт может работать на нем совсем не так, как на новом интегрированном Web-сервере ASP.NET Development Server.

Создав новый проект Web-сайта, щелкните правой кнопкой мыши на файле `Default.aspx` и удалите его. Мы скоро создадим свою собственную страницу по умолчанию.

Создание дизайна сайта

Создать мастер-страницу с общим (совместно используемым) дизайном сайта не так уж сложно, когда уже есть предварительный эскиз (или несколько эскизов, если вы делали их отдельно). Грубо говоря, нужно вырезать логотип и другие графические объекты и поместить их на HTML-страницу. Другие части схемы вроде строки меню, колонок и нижнего колонтитула, можно запросто воспроизвести при помощи таких HTML-элементов, как `DIV`. На рис. 2.4 показан шаблон, предоставленный компанией TemplateMonster (который я немного изменил и расширил).

Рис. 2.4. Первоначальный шаблон главной страницы сайта

Из этого рисунка можно вырезать сразу всю строку верхнего колонтитула и разместить на ее месте какие-нибудь DIV-контейнеры: один для ссылок меню, один для поля регистрации и еще один для переключателя темы (выпадающего списка, содержащего названия доступных тем). Эти DIV-контейнеры будут использовать абсолютное позиционирование для того, чтобы вы могли разместить ими именно там, где хотите. Определить правильные координаты левой верхней и правой верхней позиций для них довольно легко — нужно просто навести курсор на это открытое в графическом редакторе изображение и затем использовать отобразившиеся значения *x* и *y*.

Нижний колонтитул создан при помощи DIV-контейнера, содержащего фрагмент изображения шириной 1 пиксель, которое повторяется горизонтально как фон. Он еще также содержит несколько вложенных DIV-контейнеров: один — для ссылок меню (таких же, как и в меню верхнего колонтитула), а второй — для кое-какой информации об авторских правах.

И, наконец, есть еще область содержимого страницы, поделенная на три колонки. Центральная колонка имеет левое и правое поле размером 200 пикселей, которые заполняются двумя другими DIV-контейнерами, пристыкованными к границам страницы с абсолютным позиционированием. На рис. 2.5 наглядно иллюстрируются концепции, которые применены к изображению, показанному на рис. 2.4.

Создание мастер-страницы

> В этой книге предполагается, что читатель уже до определенной степени знаком с ASP.NET и Visual Studio .NET, а именно, что у него имеются практические навыки работы с какой-нибудь любой из предыдущих версий Visual Studio .NET. Поэтому основное внимание здесь уделяется новым изменениям, появившимся в версии 2.0, а не всем мельчайшим деталям. Тем, кому трудно дается выполнения описываемых шагов, следует дополнительно воспользоваться какой-нибудь посвященной ASP.NET книгой для начинающих.

Создав проект Web-сайта, как описывалось выше, создайте новый файл мастер-страницы (чтобы сделать это, выберите в меню Website (Web-сайт) пункт Add New Item⇨Master Page (Добавить новый элемент⇨Мастер-страницв)), назовите его Template.master и затем используйте визуальной конструктор, чтобы добавить на его поверхность серверные элементы управления ASP.NET и статические HTML-элементы. Однако работая с DIV-контейнерами и отдельными файлами таблиц стилей, я обнаружил, что визуальный дизайнер не дает мне той гибкости, какую я хочу. Поэтому я предпочитаю работать прямо в режиме исходного кода (Source View) и писать код вручную. Как я уже упоминал ранее, процесс создания мастер-страницы не очень отличается от процесса создания обычной страницы; самыми главными отличиями являются добавление директивы @Master в начале файла и наличие элементов ContentPlaceHolder в тех местах, где .aspx-страницы будут вставлять свое собственное содержимое.

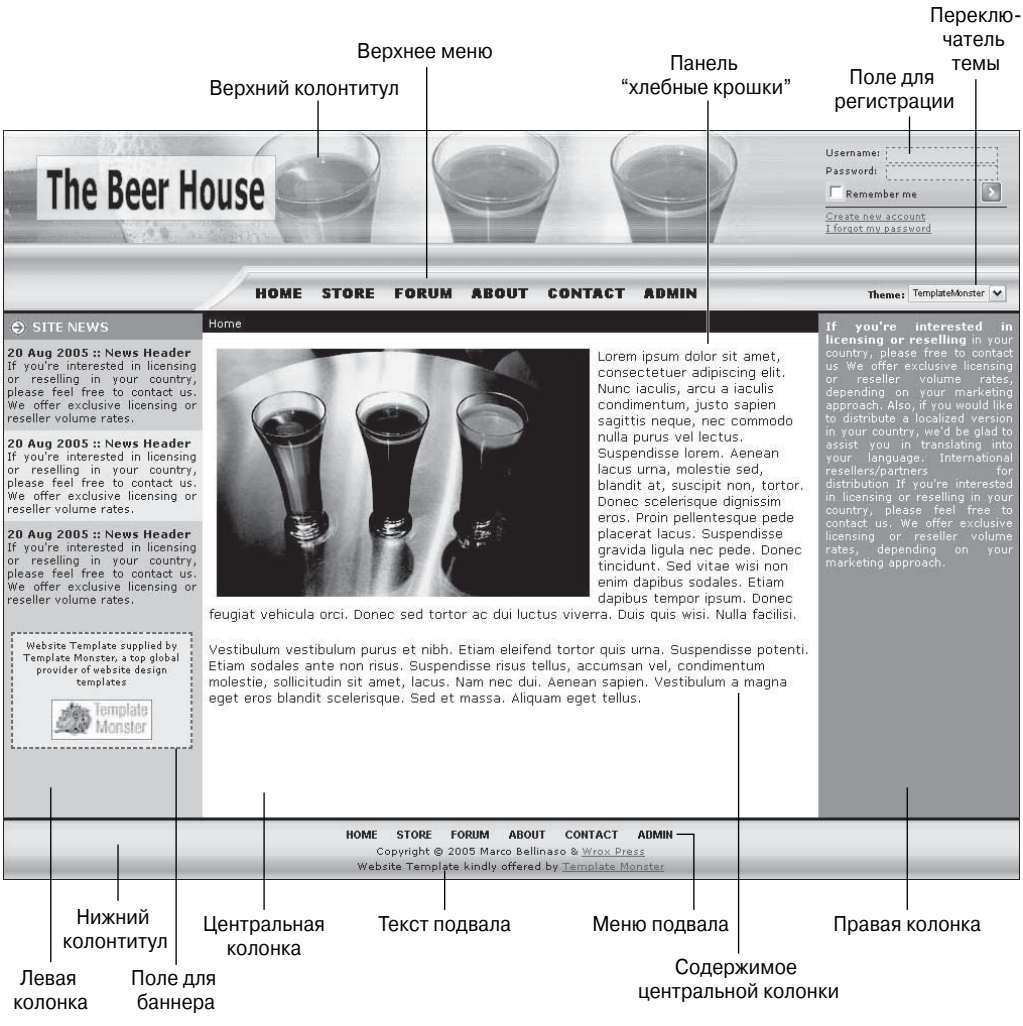

Рис. 2.5. Объяснение шаблона главной страницы сайта

Ниже приведен код, который определяет стандартные метадескрипторы HTML и верхний колонтитул сайта для файла `Template.master`.

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="Template.master.cs"
Inherits="TemplateMaster" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
  <title>TheBeerHouse</title>
</head>
```

```
<body>
  <form id="Main" runat="server">
    <div id="header">
      <div id="header2">
        <div id="headermenu">
          <asp:SiteMapDataSource ID="SiteMapDataSource1"
            runat="server" StartingNodeOffset="0" />
          <asp:Menu ID="mnuHeader" runat="server"
            CssClass="headermenulink"
            DataSourceID="SiteMapDataSource1"
            Orientation="Horizontal"
            MaximumDynamicDisplayLevels="0"
            SkipLinkText=""
            StaticDisplayLevels="2" />
    </div>
  </div>
  <div id="loginbox">Login box here...</div>
  <div id="themeselector">Theme selector here...</div>
</div>
```

Как видите, в этом первом фрагменте кода нет ничего, что бы касалось самого внешнего вида верхнего колонтитула. А все дело в том, что внешний вид контейнеров, текста и других объектов мы будем задавать в файлах стилей и обложки. Контейнер `"loginbox"` пока останется пустым; мы заполним его позже, когда доберемся до главы 4, посвященной системе безопасности и системе членства. Поле `"themeselector"` мы заполним чуть позже в этой главе, как только разработаем элемент управления, отображающий доступные стили и позволяющий пользователям делать выбор. DIV-контейнер `"headermenu"` содержит элемент управления `SiteMapPathDataSource`, который загружает содержимое файла `Web.sitmap` (который мы вскоре создадим). А также он содержит элемент управления `Menu`, который использует элемент управления `SiteMapPathDataSource` в качестве источника данных об элементах, которые нужно создать.

Далее потребуется написать DIV-контейнеры для центральной части страницы, состоящей из трех колонок.

```
<div id="container">
  <div id="container2">
    <div id="rightcol">
      <div class="text">Some text...</div>
      <asp:ContentPlaceHolder ID="RightContent" runat="server" />
  </div>
  <div id="centercol">
    <div id="breadcrumb">
       <asp:SiteMapPath ID="SiteMapPath1" runat="server" />
    </div>
    <div id="centercolcontent">
      <asp:ContentPlaceHolder ID="MainContent" runat="server">
        <p> </p><p> </p><p> </p><p> </p>
        <p> </p><p> </p><p> </p><p> </p>
      </asp:ContentPlaceHolder>
    </div>
  </div>
</div>
```

```
  <div id="leftcol">
    <div class="sectiontitle">
      <asp:Image ID="imgArrow1" runat="server"
          ImageUrl="~/images/arrowr.gif"
          ImageAlign="left" hspace="6" />Site News
    </div>
    <div class="text"><b>20 Aug 2005 :: News Header</b><br />
      News text...
    </div>
    <div class="alternatetext"><b>20 Aug 2005 :: News Header</b><br />
      Other news text...
    </div>
    <asp:ContentPlaceHolder ID="LeftContent" runat="server" />
    <div id="bannerbox">
      <a href="http://www.templatemonster.com" target="_blank">
      Website Template supplied by Template Monster,
      a top global provider of website design templates<br /><br />
      <asp:Image runat="server" ID="TemplateMonsterBanner"
        ImageUrl="~/images/templatemonster.jpg" Width="100px" />
      </a>
    </div>
  </div>
</div>
```

Обратите внимание на то, что в показанном выше коде определены три элемента управления `ContentPlaceHolder`, по одному для каждой колонки. Это позволит странице содержимого добавлять текст в трех разных позициях. Также учтите, что заполнение элемента управления `ContentPlaceHolder` каким-либо содержимым является необязательным, и в некоторых случаях у нас будут страницы, добавляющие содержимое только в центральную колонку, а для двух других колонок использующие содержимое по умолчанию, определенное в мастер-странице. Кроме того, центральная колонка также содержит вложенный `DIV`-контейнер с элементом управления `SiteMapPath` для системы навигации типа “хлебные крошки”.

В остальной части мастер-страницы определяется контейнер для нижнего колонтитула с вложенными контейнерами (подконтейнерами) для отображаемого в нижнем колонтитуле меню (которое в точности повторяет меню из верхнего колонтитула, за исключением применяемого к нему стиля) и информации об авторских правах.

```
  <div id="footer">
    <div id="footermenu">
      <asp:Menu ID="mnuFooter" runat="server"
        style="margin-left:auto; margin-right:auto;"
        CssClass="footermenulink"
        DataSourceID="SiteMapDataSource1"
        Orientation="Horizontal"
        MaximumDynamicDisplayLevels="0"
        SkipLinkText=""
        StaticDisplayLevels="2" />
    </div>
    <div id="footertext">
      <small>Copyright &copy; 2005 Marco Bellinaso &amp
      <a href="http://www.wrox.com" target="_blank">Wrox Press</a><br />
      Website Template kindly offered by
```

```
      <a href="http://www.templatemonster.com" target="_blank">
        Template Monster</a></small>
    </div>
  </div>
</form>
</body>
</html>
```

Примечание по поводу межбраузерной совместимости. *DIV-контейнер, объявленный в показанном выше коде, будет выравнивать текст по центру. Однако элемент управления* `Menu`*, объявленный внутри него, во время выполнения будет визуализироваться как HTML-таблица, а таблицы не воспринимаются как текст браузерами типа Firefox и, следовательно, не выравниваются как текст. Поскольку мы ориентируемся не только на браузер Internet Explorer, но и на Firefox, мы должны удостовериться в том, что эта таблица будет выравниваться по центру в обоих этих браузерах, поэтому я добавил в объявление элемента управления* `Menu` *атрибут* `style`*, чтобы разместить одинаковое поле слева и справа таблицы меню, причем насколько возможно большое. Благодаря этому таблица будет выравниваться по горизонтали. Атрибут* `style` *не отображается ни на какое серверное свойство, предоставляемое этим элементом управления; оказывается, что такого свойства и свойства, которое бы во время выполнения интерпретировалось как "выравнивание", просто нет, поэтому я воспользовался этим HTML-атрибутом. Поскольку он не отображается на свойство элемента управления, он будет просто прикрепляться "в таком, как он есть" виде к HTML-таблице, генерируемой во время визуализации элемента управления.*

Создание файла карты сайта

Зная, как легко добавляются, удаляются и изменяются ссылки в меню сайта, когда используется файл `sitemap` и элемент управления `SiteMapPath`, на этом этапе волноваться о ссылках вы не должны. Информация, касающаяся ссылок, может вводиться и позже. Сейчас для примера вы можете добавить в файл `sitemap` несколько предварительных ссылок, но помните, что вы всегда сможете вернуться и изменить его, если возникнет такая необходимость. Итак, добавьте в проект файл `Web.sitemap` (выбрав в меню Website (Web-сайт) пункт Add New Item⇨Site Map (Добавить новый элемент⇨Карта сайта)), а затем добавьте уже внутри этого файла следующие XML-узлы.

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode title="Home" url="~/Default.aspx">
    <siteMapNode title="Store" url="~/Store/Default.aspx">
      <siteMapNode title="Shopping cart" url="~/Store/ShoppingCart.aspx" />
    </siteMapNode>
    <siteMapNode title="Forum" url="~/Forum/Default.aspx" />
    <siteMapNode title="About" url="~/About.aspx" />
    <siteMapNode title="Contact" url="~/Contact.aspx" />
    <siteMapNode title="Admin" url="~/Admin/Default.aspx" />
  </siteMapNode>
</siteMap>
```

Вам наверное интересно, почему узел `Home` выступает в роли корневого узла для всех остальных и не находится с ними на одном уровне. Это конечно не обязательно, но я хочу, чтобы элемент управления `SiteMapPath` всегда показывал ссылку Home (Домой) перед остальной частью пути, ведущего к текущей странице, а для этого

Home обязательно должен быть корневым узлом. На самом деле, элемент управления SiteMapPath не запоминает предыдущие страницы — он просто отыскивает в карте сайта XML-узел, описывающий текущую страницу, и отображает ссылки на все его родительские узлы.

> Если проект разворачивается в корневой папке IIS-сайта, корневая папка может обозначаться как /. Однако если проект сайта разворачивается в какой-нибудь виртуальной подпапке, такой вариант уже не подходит. Как видите, в показанном выше файле sitemap для обозначения корневой папки используется комбинация символов "~/". Эти URL-адреса будут разрешаться во время выполнения, в соответствии с тем, где находятся страницы, и этот процесс будет выполняться одинаково хорошо как в случае, когда страницы находятся в корневой папке сайта, так и в случае, если они находятся в виртуальной подпапке.
>
> Когда используется интегрированный Web-сервер, он всегда запускает Web-сайт так, будто бы тот разворачивался в виртуальной папке и его URL-адрес включает имя проекта. А это значит, что вы должны использовать в своих URL-адресах комбинацию "~/", чтобы свести к минимуму количество проблем во время разворачивания.

Создание первой темы

Пришло время создать первую тему для мастер-страницы: TemplateMonster. Существует два способа сделать это, которые с функциональной точки зрения являются абсолютно эквивалентными. Первый — это самостоятельно создать в проекте новую папку с именем App_Themes и затем добавить в нее новую подпапку с именем TemplateMonster. Второй — прибегнуть к помощи VS2005: выбрать в меню Website (Web-сайт) пункт Add Folder⇨Theme Folder (Добавить папку⇨Папка тем) и назвать появившуюся после этого папку TemplateMonster (папку App_Themes в этом случае создавать не придется, VS2005 создаст ее автоматически). Папка App_Themes является специальной, потому что использует зарезервированное имя и отображается в окне проводника решений (Solution Explorer) на сером фоне. Далее нужно выбрать папку App_Themes\TemplateMonster и добавить в нее файл таблицы стилей, что можно сделать следующим образом: выбрать в меню Website (Web-сайт) пункт Add New Item⇨Stylesheet (Добавить новый элемент⇨Таблица стилей) и назвать появившийся после этого файл Default.css. Имя, которое дается CSS-файлу, не имеет значения, потому что абсолютно все файлы, находящиеся в папке текущей темы, все равно будут автоматически связываться страницей .aspx во время выполнения.

Для вашего удобства следующий далее код включает часть классов стилей, определенных в этом файле (полную версию таблицы стилей вы сможете найти в загружаемом коде).

```
body
{
  margin: 0px; font-family: Verdana; font-size: 12px;
}

#container
{
  background-color: #818689;
}
```

```
#container2
{
  background-color: #bcbfc0; margin-right: 200px;
}

#header
{
  padding: 0px; margin: 0px; width: 100%; height: 184px;
  background-image: url(images/HeaderSlice.gif);
}

#header2
{
  padding: 0px; margin: 0px; width: 780px; height: 184px;
  background-image: url(images/Header.gif);
}

#headermenu
{
  position: relative; top: 153px; left: 250px; width: 500px;
  padding: 2px 2px 2px 2px;
}

#breadcrumb
{
  background-color: #202020; color: White; padding: 3px; font-size: 10px;
}

#footermenu
{
  text-align: center; padding-top: 10px;
}

#loginbox
{
  position: absolute; top: 16px; right: 10px; width: 180px; height: 80px;
  padding: 2px 2px 2px 2px; font-size: 9px;
}

#themeselector
{
  position: absolute; text-align: right; top: 153px; right: 10px; width: 180px;
  height: 80px; padding: 2px 2px 2px 2px; font-size: 9px;
}

#footer
{
  padding: 0px; margin: 0px; width: 100%; height: 62px;
  background-image: url(images/FooterSlice.gif);
}

#leftcol
{
  position: absolute; top: 184px; left: 0px; width: 200px;
  background-color: #bcbfc0; font-size: 10px;
}

#centercol
{
```

```
  position: relative inherit; margin-left: 200px; padding: 0px;
  background-color: white; height: 500px;
}

#centercolcontent
{
  padding: 15px 6px 15px 6px;
}

#rightcol
{
  position: absolute; top: 184px; right: 0px; width: 198px; font-size: 10px;
  color: White; background-color: #818689;
}
.footermenulink
{
  font-family: Arial; font-size: 10px; font-weight: bold;
  text-transform: uppercase;
}

.headermenulink
{
  font-family: Arial Black; font-size: 12px; font-weight: bold;
  text-transform: uppercase;
}

/* остальные стили были опущены для краткости */
```

Обратите внимание на то, как определенные элементы (такие как "loginbox", "themeselector", "leftcol" и "rightcol") используют абсолютное позиционирование. Также обратите внимание на то, что здесь присутствует целых два контейнера с двумя разными стилями для верхнего колонтитула. Первый, header, в ширину занимает всю страницу (если явная ширина не указывается и абсолютное позиционирование не используется, DIV всегда будет иметь неявную ширину, которая равна 100%), и имеет фоновый рисунок размером в 1 пиксель, который (неявно, по умолчанию) повторяется в горизонтальном направлении. Второй, header2, по ширине соответствует рисунку Header.gif, который он использует в качестве фона, и размещается поверх первого контейнера. В результате этого получается, что первый контейнер служит для продолжения фона второго контейнера, который имеет фиксированную ширину. Два контейнера нам необходимы только потому, чтобы мы хотим иметь динамическую схему, которая будет полностью заполнять страницу по ширине. Если бы мы хотели использовать схему фиксированной ширины, нам бы вполне хватило и одного контейнера.

Все изображения, указанные в этом файле таблицы стилей, находятся в папке Images внутри папки App_Themes/TemplateMonster. То есть все объекты, из которых состоит тема, хранятся вместе.

Теперь добавьте файл обложки под названием Controls.skin. (Чтобы сделать это, выберите папку TemplateMonster, а затем выберите в меню Website (Web-сайт) пункт Add New Item⇨Skin File (Добавить новый элемент⇨Файл обложки)). Вы поместите в этот файл все серверные стили, которые должны применяться к элементам управления всех типов. Конечно, вы могли бы создать отдельный файл для каждого элемента управления, но, на мой взгляд, управлять стилями в одном файле гораздо легче.

Следующий далее код содержит две безымянные обложки для элементов управления TextBox и SiteMapPath и две именованных (SkinID) обложки для элемента управления Label.

```
<asp:TextBox runat="server" BorderStyle="dashed" BorderWidth="1px" />
<asp:Label runat="server" SkinID="FeedbackOK" ForeColor="green" />
<asp:Label runat="server" SkinID="FeedbackKO" ForeColor="red" />

<asp:SiteMapPath runat="server">
  <PathSeparatorTemplate>
    <asp:Image runat="server" ImageUrl="images/sepwhite.gif"
      hspace="4" align="middle" />
  </PathSeparatorTemplate>
</asp:SiteMapPath>
```

Первые три обложки присутствуют здесь в основном для демонстрационных целей, поскольку точно таких же результатов можно добиться и путем определения обычных CSS-стилей. Обложку для элемента управления SiteMapPath, однако, повторить при помощи CSS-стилей далеко не так просто, потому что этот элемент управления не отображается ни на какой один HTML-элемент. В показанном выше коде эта обложка указывает, *что* должно применяться в качестве разделителя для ссылок, ведущих к текущей странице — а именно, что использоваться должно изображение, представляющее стрелку.

Создание примера страницы Default.aspx

Теперь, когда у вас есть готовая мастер-страница и тема, вы можете протестировать их, создав пробную страницу содержимого. Чтобы сделать это, сначала добавьте в проект новую Web-страницу под названием Default.aspx (выделив проект в окне проводника решений Solution Explorer и затем выбрав в меню Website (Web-сайт) пункт Add New Item⇨Web Form (Добавить новый элемент⇨Web-форма)), отметьте флажок Select Master Page (Выбрать мастер-страницу) в диалоговом окне Add New Item (Добавление нового элемента) и в появившемся после этого втором диалоговом окне выберите мастер-страницу, которая должна использоваться, а именно — страницу Template.master. После того как вы это сделаете, страница будет содержать только элементы управления Content, соответствующие элементам управления ContentPlaceHolder; дескрипторов <html>, <body>, <head> и <form>, которые обязательно присутствовали бы в противном случае (т.е. если бы вы не указали мастер-страницу), не будет. Далее вы можете добавить в центральный элемент управления ContentPlaceHolder какое-нибудь содержимое, как показано ниже.

```
<%@ Page Language="C#" AutoEventWireup="true" MasterPageFile="~/Template.master"
CodeFile="Default.aspx.cs" Inherits="_Default" Title="The Beer House" %>

<asp:Content ID="Content1" ContentPlaceHolderID="RightContent" Runat="Server">
</asp:Content>

<asp:Content ID="MainContent" runat="server" ContentPlaceHolderID="MainContent">
  <asp:Image ID="imgBeers" runat="server"
    ImageUrl="~/Images/3beers.jpg" ImageAlign="left" hspace="8" />
  Lorem ipsum dolor sit amet, consectetuer adipiscing elit...
</asp:Content>

<asp:Content ID="Content3" ContentPlaceHolderID="LeftContent" Runat="Server">
</asp:Content>
```

Вы также могли бы добавить атрибут Theme в директиву @Page, присвоив ему значение "Template Monster". Однако вместо того, чтобы делать это здесь, вы можете сделать это в файле web.config один единственный раз, и тем самым применить эту настройку ко всем страницам. Выделите проект в окне проводника Solution Explorer и затем выберите в меню Website (Web-сайт) пункт Add New Item⇨Web Configuration File (Добавить новый элемент⇨Конфигурационный файл Web). Удалите атрибут MasterPageFile из кода страницы Default.aspx, потому вы его тоже поместите в файл web.config, как показано ниже.

```
<?xml version="1.0"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <pages theme="TemplateMonster" masterPageFile="~/Template.master" />
    <!-- здесь идут остальные параметры... -->
  </system.web>
</configuration>
```

Зачем выбирать мастер-страницу в диалоговом окне New Item (Новый элемент) при создании страницы Default.aspx, если потом атрибут мастер-страницы все равно нужно сразу же удалить? А потому, что только в этом случае VS2005 создаст необходимые элементы управления Content, а не HTML-код обычной страницы.

Создание второй темы

Для того чтобы протестировать функциональную возможность, позволяющую пользователям выбирать различные темы, нам необходимо иметь более чем одну тему. Поэтому под папкой App_Themes создайте еще одну папку по имени PlainHtmlYellow (для этого выделите проект, щелкните правой кнопкой мыши, во всплывающем меню выберите пункт Add Folder⇨Theme Folder (Добавить папку⇨Папку тем) и присвойте появившейся после этого папке имя PlainHtmlYelLow), а потом скопируйте и вставьте в нее весь файл Default.css из папки TemplateMonster, изменив его так, чтобы он выглядел по-другому. В рассматриваемом примере я изменил почти все контейнеры так, чтобы не использовался никакой фоновый рисунок и чтобы верхний и нижний колонтитул заполнялись простыми однотонными цветами, такими же, как левая и правая колонки. Кроме того, я еще изменил размер некоторых элементов, а также их позицию. В частности, я полностью поменял местами позиции левой и правой колонки (которые используют абсолютное позиционирование), т.е. сделал так, чтобы контейнер leftcol пристыковывался к правой границе, а контейнер rightcol — к левой. Все это я смог выполнить, просто изменив несколько классов стилей, как показано ниже.

```
#leftcol
{
  position: absolute;
  top: 150px;
  right: 0px;
  width: 200px;
  background-color: #ffb487;
  font-size: 10px;
}
```

```
#rightcol
{
  position: absolute;
  top: 150px;
  left: 0px;
  width: 198px;
  color: White;
  background-color: #8d2d23;
  font-size: 10px;
}
```

Вот она мощь DIV-контейнеров и таблиц стилей: изменяешь парочку стилей, и содержимое, которые было слева, перемещается вправо. Это был довольно простой пример; вы же можете пойти гораздо дальше и создавать полностью отличающиеся схемы, в одном случае со скрытыми частями, в другом — с частями большего размера, и т.д.

Что касается файла обложки, просто скопируйте и вставьте весь файл controls.skin из папки TemplateMonster, а затем удалите из него определение для элементов управления TextBox и SiteMapPath для того, чтобы они имели стандартный внешний вид. Вы увидите отличие, когда мы изменим тему во время выполнения. Если вы потом снова захотите изменить их внешний вид, вы сможете вернуться и добавить в этот файл новое определение стиля, больше ничего не меняя.

Создание пользовательского элемента управления ThemeSelector

Теперь у вас есть мастер-страница и несколько тем для нее, так что вы можете приступать к созданию пользовательского элемента управления, который будет отображать список доступных тем и позволять пользователю выбирать одну из них. Создав этот элемент управления, вы должны будете встроить его в мастер-страницу, а именно — в DIV-контейнер "themeselector". Прежде чем создавать его, создайте новую папку под названием "Contols", в которую вы будете помещать все свои пользовательские элементы управления для того, чтобы они хранились отдельно от страниц, тем самым улучшая организацию (для этого выделите проект, щелкните правой кнопкой мыши, во всплывающем меню выберите пункт Add Folder⇨Regular Folder (Добавить папку⇨Обычная папка) и присвойте появившейся после этого папке имя Controls). Чтобы создать новый пользовательский элемент управления, щелкните правой кнопкой мыши на папке Controls, во всплывающем меню выберите пункт Add New Item⇨Web User Control (Добавить новый элемент⇨Пользовательский элемент управления Web) и назовите появившийся после этого файл ThemeSelector.ascx. Содержимое этого .ascx-файла будет выглядеть очень просто и включать только строку и элемент управления DropDownList:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeFile="ThemeSelector.ascx.cs"
Inherits="ThemeSelector" %>
<b>Theme:</b>
<asp:DropDownList runat="server" ID="ddlThemes" AutoPostBack="true" />
```

Обратите внимание на наличие у выпадающего списка свойства AutoPostBack, для которого установлено значение true, означающее, что данная страница будет автоматически отправляться серверу в случае изменения пользователем выбранного значения. Как вы скоро увидите, процедура заполнения этого выпадающего списка

названиями доступных тем и процедура загрузки выбранной темы на самом деле будут выполняться в файле code-behind этого элемента управления и в базовом классе страницы. В файле code-behind вы должны заполнить этот выпадающий список массивом строк, возвращаемым вспомогательным методом (`Helper`), и затем выбрать элемент, который имеет такое же значение текущей страницы `Theme`.

```
public partial class ThemeSelector : System.Web.UI.UserControl
{
  protected void Page_Load(object sender, EventArgs e)
  {
    ddlThemes.DataSource = Helpers.GetThemes();
    ddlThemes.DataBind();

    ddlThemes.SelectedValue = this.Page.Theme;
  }
}
```

Метод `GetThemes` определяется в файле `Helpers.cs`, который находится в еще одной специальной папке под названием `App_Code`. Файлы в этой папке автоматически компилируются во время выполнения механизмом ASP.NET, так что компилировать их перед запуском проекта не требуется. Вы даже можете изменять эти файлы исходного кода C# во время работы приложения и щелкать на кнопке Refresh (Обновить): измененный файл будет компилироваться заново в новой временной сборке и загружаться. Более подробно о новой модели компиляции будет рассказываться чуть позже в этой книге, а именно — в главе 12, посвященной процессу развертывания.

Метод `GetThemes` использует метод `GetDirectories` класса `System.IO.Directory` для извлечения массива с путями всех папок, содержащихся в папке `~/App_Themes` (этот метод ожидает получить физический путь, а не URL-адрес — однако физический путь, на который указывает URL-адрес, может быть извлечен при помощи метода `Server.MapPath`). Возвращаемый массив строк содержит целый путь, а не одно имя папки, так что вы должны прогнать этот массив по циклу и перезаписать каждый элемент соответствующим в этом пути именно ему значением (которое возвращает статический метод `System.IO.Path.GetFileName`). После первого заполнения массив сохраняется в кэше ASP.NET, так что следующие запросы будут извлекать его уже оттуда, и, следовательно, делать это гораздо быстрее. Следующий код иллюстрирует все содержимое класса `Helpers` (`App_Code/Helpers.cs`).

```
namespace MB.TheBeerHouse.UI
{
  public static class Helpers
  {
    /// <summary>
    /// Возвращает массив с именами всех локальных тем
    /// </summary>
    public static string[] GetThemes()
    {
      if (HttpContext.Current.Cache["SiteThemes"] != null)
      {
        return (string[])HttpContext.Current.Cache["SiteThemes"];
      }
      else
      {
```

```
            string themesDirPath =
               HttpContext.Current.Server.MapPath("~/App_Themes");
            // извлекаем массив папок тем из каталога /app_themes
            string[] themes = Directory.GetDirectories(themesDirPath);
            for (int i = 0; i <= themes.Length - 1; i++)
            themes[i] = Path.GetFileName(themes[i]);
            // кэшируем массив с зависимостью от папки
            CacheDependency dep = new CacheDependency(themesDirPath);
            HttpContext.Current.Cache.Insert("SiteThemes", themes, dep);
            return themes;
          }
        }
      }
    }
```

Теперь, когда у вас есть необходимый элемент управления, вернитесь к мастер-странице и добавьте в начале ее файла, в режиме Source View, следующую строку, чтобы сослаться на внешний пользовательский элемент управления:

```
<%@ Register Src="Controls/ThemeSelector.ascx" TagName="ThemeSelector"
TagPrefix="mb" %>
```

Далее объявите экземпляр элемента управления в том месте, где он должен находиться, а именно — внутри контейнера "themeselector":

```
<div id="themeselector">
  <mb:ThemeSelector id="ThemeSelector1" runat="server" />
</div>
```

Код, обрабатывающий переключение на новую тему, нельзя разместить в событии SelectedIndexChanged элемента управления DropDownList, потому что оно происходит в жизненном цикле страницы слишком поздно. Как уже говорилось, в разделе "Проект", новая тема должна применяться в таком событии страницы как PreInit. Также вместо того, чтобы писать этот код для каждой страницы, мы напишем его только один раз в специальной базовой странице. Наша задача — прочитать значение выбранного индекса DropDownList из нашего специального базового класса и затем применить тему, указанную в элементе управления DropDownList. Однако получать доступ к элементам управления и их значениям из обработчика события PreInit нельзя, потому что оно все-таки происходит в жизненном цикле страницы слишком рано. Следовательно, значение этого элемента управления нужно считать в каком-нибудь происходящем позже событии сервера: событие Load является прекрасным местом для его считывания.

Тем не менее, находясь в обработчике события Load, вы не будете знать идентификатора элемента управления DropDownList, а это значит, что вам будет нужен какой-нибудь способ, позволяющий идентифицировать этот элемент управления; тогда вы сможете считать его значение путем получения доступа к необработанным данным, которые были отправлены обратно серверу, через коллекцию Request.Form. Но все равно остается еще одна проблема: чтобы извлечь значение элемента управления из этой коллекции, нужно знать его идентификатор, а этот идентификатор может быть разным, что зависит от контейнера, в который вы его помещаете, и жестко закодировать его — тоже не лучший вариант, потому что в будущем у вас вполне может возникнуть желание изменить его местонахождение. Взамен, когда этот элемент управления создается впервые, вы можете сохранить его клиентский идентификатор в стати-

ческом поле класса, так чтобы он поддерживался на протяжении всего жизненного цикла приложения между различными запросами (операциями обратной отправки данных) до тех пор, пока приложение не завершит свою работу (а точнее, до тех пор, пока не загрузится домен приложения со всеми его частями). Поэтому сейчас добавьте в папку App_Code файл Globals.cs и напишите внутри него следующий код:

```
namespace MB.TheBeerHouse
{
  public static class Globals
  {
    public static string ThemesSelectorID = "";
  }
}
```

Далее вернитесь к файлу code-behind элемента управления ThemeSelector и добавьте в него следующий код, чтобы сохранить его идентификатор в статическом поле:

```
public partial class ThemeSelector : System.Web.UI.UserControl
{
  protected void Page_Load(object sender, EventArgs e)
  {
    if (Globals.ThemesSelectorID.Length == 0)
      Globals.ThemesSelectorID = ddlThemes.UniqueID;

    ddlThemes.DataSource = Helpers.GetThemes();
    ddlThemes.DataBind();

    ddlThemes.SelectedValue = this.Page.Theme;
  }
}
```

Теперь вы можете создать для своих страниц специальный базовый класс, и это будет просто еще один обычный класс, который вы разместите под App_Code, и который будет наследоваться от System.Web.UI.Page. Вы должны перекрыть его метод OnPreInit, чтобы решить следующие задачи.

1. Проверить, является ли текущий запрос операцией обратной отправки данных (postback). Если да, был ли он вызван выпадающим списком ThemeSelector. Как и в ASP.NET 1.x, все страницы с серверной формой имеют скрытое поле "_EVENTTARGET", которое будет устанавливаться с идентификатором HTML-элемента управления, инициирующего обратную отправку данных (если только это не кнопка Submit (Отправить)). Чтобы удостовериться в соблюдении этого условия, вы можете просто проверить, содержит ли элемент "_EVENTTARGET" коллекции Form идентификатор выпадающего списка, на основании идентификатора, считываемого из класса Globals.
2. Если условия, перечисленные в пункте 1, соблюдены, вы должны извлечь имя выбранной темы из элемента коллекции Form со значением Id равным идентификатору, сохраненному в Globals, и использовать его для установки значения для свойства Theme страницы. Затем вы также должны сохранить это значение в переменной Session. Это нужно сделать для того, чтобы последующие запросы, поступающие от того же пользователя, правильно загружались в новую выбранную им тему и не приводили к возврату к стандартной (используемой по умолчанию) теме.

3. Если текущей запрос не является операцией обратной отправки данных, вы должны проверить, содержится ли в переменной Session, использовавшейся в пункте 2, какое-нибудь значение или нет. Если да, вы должны извлечь это значение и использовать его для свойства Theme страницы.

Следующий фрагмент превращает это описание в реальный код:

```
namespace MB.TheBeerHouse.UI
{
  public class BasePage : System.Web.UI.Page
  {
    protected override void OnPreInit(EventArgs e)
    {
      string id = Globals.ThemesSelectorID;
      if (id.Length > 0)
      {
        // если это операция обратной отправки, вызванная выпадающим списком
        // переключателя тем (ThemeSelector), извлекаем выбранную тему
        // и используем ее для текущего запроса страницы
        if (this.Request.Form["__EVENTTARGET"] == id &&
          !string.IsNullOrEmpty(this.Request.Form[id]))
        {
          this.Theme = this.Request.Form[id];
          this.Session["CurrentTheme"] = this.Theme;
        }
        else
        {
          // если это не операция обратной отправки, или операция обратной
          // отправки, вызванная не переключателем тем (ThemeSelector), а
          // каким-нибудь другим элементом управления, устанавливаем для темы
          // страницы значение, содержащееся в Session, если оно там есть
          if (this.Session["CurrentTheme"] != null)
            this.Theme = this.Session["CurrentTheme"].ToString();
        }
      }

      base.OnPreInit(e);
    }
  }
}
```

Недостатком показанного здесь подхода является то, что выбранная тема сохраняется в переменной сеанса (Session), которая очищается при завершении сеанса, т.е. тогда, когда пользователь закрывает окно браузера или когда он не обращается к странице в течение 20 минут (длительность можно регулировать). Намного более удачным решением было бы использовать свойства Profile, которые помимо прочих своих преимуществ также умеют сохранять свои значения между сеансами. Вы ознакомитесь с этой новой функциональной возможностью ASP.NET 2.0 и измените данный код так, чтобы он использовал ее, в главе 4.

Последнее, что вы должны сделать — это изменить используемый по умолчанию файл code-behind для страницы Default.aspx так, чтобы вместо стандартного класса Page он работал с вашим собственным классом BasePage. Ваш специальный базовый класс, в свою очередь, будет вызывать исходный класс Page. Чтобы сделать это, вам

нужно изменить всего лишь одно слово, а именно — изменить слово `Page` на слово `BasePage`, как показано ниже.

```
public partial class _Default : MB.TheBeerHouse.UI.BasePage
{
  protected void Page_Load(object sender, EventArgs e)
  { }
}
```

Все готово! Если вы сейчас запустите проект, то по умолчанию увидите домашнюю страницу, показанную на рис. 2.4 (за исключением поля для регистрации, которое пока что ничего не содержит — мы заполним его в главе 5), с примененной к ней темой `TemplateMonster`. Если вы в выпадающем списке `ThemeSelector` выберете элемент `PlainHtmlYellow`, внешний вид домашней страницы изменится, и она станет выглядеть так, как показано на рис. 2.6.

Рис. 2.6. Домашняя страница с выбранной темой `PlainHtmlYellow`

Еще один стилистический штрих

Одной страницы (`Default.aspx`) не достаточно для того, чтобы протестировать все, о чем мы рассказывали и реализовывали в этой главе. Например, мы пока так и не увидели, как же работает элемент управления `SiteMapPath`, а все потому, что для того чтобы он показал хоть какую-нибудь ссылку, нужно перейти с домашней страницы на какую-нибудь другую. Вы можете запросто реализовать страницы `Contact.aspx` и `About.aspx`, если хотите протестировать его. Лично я для примера возьму страницу `Contact.aspx`, так как я хочу добавить еще несколько дополнительных стилисти-

ческих штрихов в тему TemplateMonster, чтобы она еще больше отличалась от темы PlainHtmlYellow. В конечном итоге эта страница будет выглядеть так, как показано на рис. 2.7.

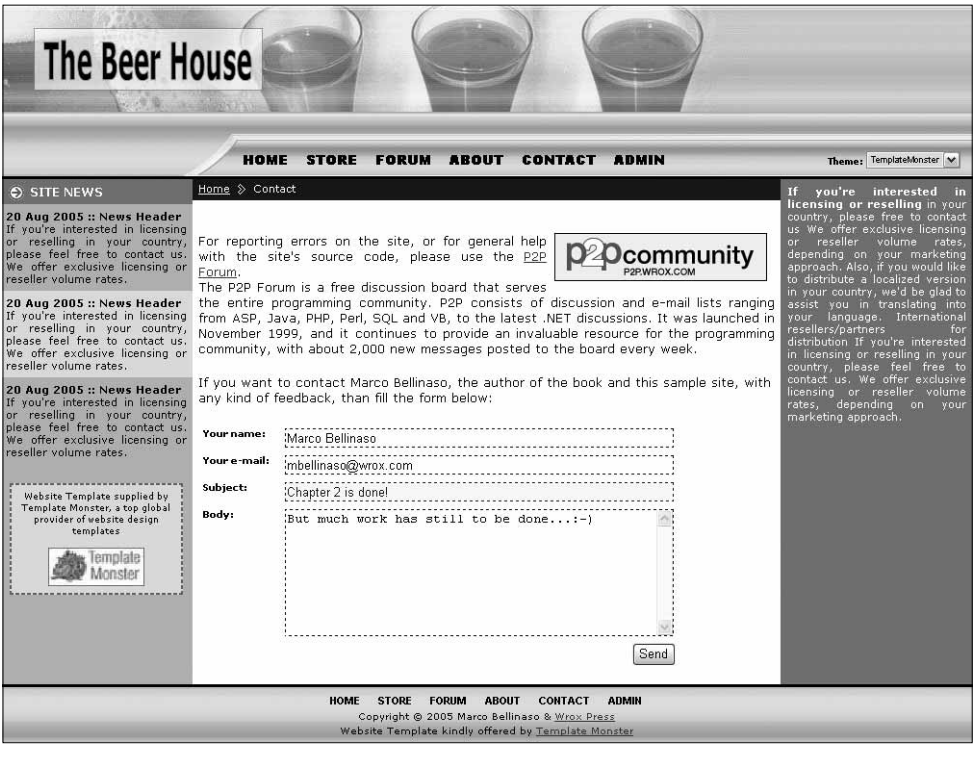

Рис. 2.7. Домашняя страница с выбранной темой TemplateMonster

Приводить код, который управляет работой этой страницы и отправляет сообщения, в данной главе я не буду, потому что этому посвящена следующая глава. Я также не буду показывать вам содержимое файла .aspx, поскольку это просто элемент управления Content, внутри которого находится несколько параграфов текста и несколько элементов управления TextBox с соответствующими им элементами управления типа Validator. Вместо этого я лучше обращу ваше внимание на то, что текстовое поле Subject (Тема) имеет желтый цвет фона и что это так потому, что оно представляет собой принимающий входные данные элемент управления с фокусом. Подсветка активного элемента управления может облегчать жизнь пользователям, позволяя им сразу же видеть, в каком элементе управления они находятся, что может быть не так уж очевидно, если им приходится иметь дело с множеством элементов управления, которые отображаются во множестве колонок и строк. Подобный эффект реализуется довольно просто: нужно просто обработать клиентские события onfocus и onblur элементов управления, принимающих входные данные, и соответственно, применить к ним CSS-стиль или удалить его путем установки атрибута className. В данном примере класс стиля указывает, что цвет фона должен быть желтым, а цвет текста — синим. Этот класс, показанный ниже, следует просто добавить в файл Default.css папки TemplateMonster:

```
.highlight
{
  background-color: #fefbd2;
  color: #000080;
}
```

Чтобы добавить обработчики для клиентских JavaScript-событий `onfocus` и `onblur`, нужно всего лишь добавить в коллекцию `Attributes` элемента управления несколько комбинаций `имя_атрибута/значение`, дабы во время выполнения они визуализировались в таком, каком они есть, виде вместе со всеми остальными атрибутами, визуализируемыми данным элементом управления по умолчанию. Еще можно добавить в созданный ранее класс `Helpers` новый статический метод, чтобы охватить весь необходимый код и вызывать его более простым образом, когда он будет нужен. Этот новый метод называется `SetInputControlsHighlight` и принимает следующие параметры: ссылку на элемент управления, имя класса стиля, который должен применяться к активному элементу управления, и булевское значение, указывающее, должна ли данная процедура распространятся только на текстовые поля (`TextBox`) или также и на элементы управления вроде `DropDownList`, `ListBox`, `RadioButton`, `CheckBox`, `RadioButtonList` и `CheckBoxList`. Если переданный элемент управления оказывается элементом управления правильного типа, этот метод добавляет в него атрибуты `onfocus` и `onblur`. В противном случае, если у него есть дочерние элементы управления, он рекурсивно вызывает сам себя. Это означает, что вы можете передать ссылку на `Page` (которая сама по себе является элементом управления, потому что наследуется от базового класса `System.Web.UI.Control`), на `Panel` или на какой-нибудь другой элемент управления типа контейнера, и тем самым также неявно передать этому методу и все дочерние элементы управления. Ниже показан полный код этого метода.

```
public static class Helpers
{
  public static string[] GetThemes() { ... }

  public static void SetInputControlsHighlight(Control container,
    string className, bool onlyTextBoxes)
  {
    foreach (Control ctl in container.Controls)
    {
      if ((onlyTextBoxes && ctl is TextBox) ||
      (!onlyTextBoxes && (ctl is TextBox || ctl is DropDownList ||
      ctl is ListBox || ctl is CheckBox || ctl is RadioButton ||
      ctl is RadioButtonList || ctl is CheckBoxList)))
      {
        WebControl wctl = ctl as WebControl;
        wctl.Attributes.Add("onfocus", string.Format(
          "this.className = '{0}';", className));
        wctl.Attributes.Add("onblur", "this.className = '';");
      }
      else
      {
        if (ctl.Controls.Count > 0)
          SetInputControlsHighlight(ctl, className, onlyTextBoxes);
      }
    }
  }
}
```

Чтобы выполнить этот код в событии Load любой страницы, нужно в созданном ранее классе BasePage перекрыть метод OnLoad, как показано ниже:

```
namespace MB.TheBeerHouse.UI
{
  public class BasePage : System.Web.UI.Page
  {
    protected override void OnPreInit(EventArgs e) { ... }

    protected override void OnLoad(EventArgs e)
    {
      // добавляем java-сценарии onfocus и onblur во все принимающие входные
      // данные элементы управления на форме для того, чтобы активный элемент
      // управления имел другой внешний вид
      Helpers.SetInputControlsHighlight(this, "highlight", false);

      base.OnLoad(e);
    }
  }
}
```

Этот код будет выполняться всегда, невзирая на то, что у темы PlainHtmlYellow нет определения для класса "highlight" в файле Default.css. В этой теме активный элемент управления не будет иметь никакого определенного стиля.

"Хитрые приемчики" вроде этого реализуются очень легко и быстро, но могут действительно очень существенно улучшать интерфейс и приятно впечатлять пользователей. Более того, простота, появляющаяся в результате использования для страниц содержимого специального базового класса, значительно упрощает реализацию будущих требований.

Резюме

В этой главе мы создали основу для такого уровня сайта, как пользовательский интерфейс. Мы спроектировали и реализовали мастер-страницу с общими HTML-элементами и графикой, тем самым обеспечив возможность быстро изменять схему и графическое оформление сайта в случае необходимости путем внесения изменений в один единственный файл. Мы также использовали темы — еще одну новую функциональную возможность, представленную в ASP.NET вместе с мастер-страницами — чтобы создать несколько разных визуальных представлений для одной и той же мастер-страницы. Мы создали механизм, позволяющий пользователям динамически выбирать понравившуюся им тему из выпадающего списка, дабы они могли изменять внешний вид нашего сайта в соответствии со своими предпочтениями и вкусами. Мы также использовали новый файл Web.sitemap и элементы управления Menu и SiteMapPath, чтобы реализовать гибкую и удобную в обслуживании систему навигации, которая тоже является новинкой ASP.NET 2.0. И, наконец, мы воспользовались специальным классом BasePage, сделав так, чтобы страницы не только выглядели одинаково, но и, в некоторых случаях, еще и вели себя тоже одинаково. В общем, мы уже разработали несколько значительных функциональных возможностей, а кода при этом написали не так уж много. Для того чтобы сделать все, что мы сделали в этой главе, при помощи ASP.NET 1.x или даже какой-нибудь еще более старой технологии, понадобятся сотни, а то и тысячи строк кода. В следующей главе мы продолжим разговор об основах, но уже о тех, что нужны для уровня бизнес-логики.