Глава 1. ASP.NET Web Forms сегодня

> Когда к вам приходит вдохновение, это прекрасно, но все остальное время писатель вынужден руководствоваться методом. Ждать вдохновения приходится слишком долго.
>
> *Леонард Бернстайн*

На первых порах своего существования веб-программирование требовало использования необычной модели программирования, а также языков и инструментов, неизвестных или малознакомых большинству программистов. В 1990-е годы каждому, кто пытался построить простейший веб-сайт, приходилось изучать синтаксис HTML и хотя бы простейшие команды и объекты JavaScript. Для формирования совершенно нового набора навыков требовались усилия и время, а программисты отвлекались от другой (возможно, более производительной) работы.

Код и пользовательский интерфейс веб-страниц (иногда объединяемые общим термином «смешанная разметка») в прошлом десятилетии приходилось записывать вручную. Так возникал своего рода рубеж, отделявший «классических» программистов C/C++/Java от эксцентричных веб-разработчиков. А все большее количество программистов, использовавших Microsoft Visual Basic, находилось посередине и в некотором смысле воздерживалось от решительных шагов в том или ином направлении — будь то серверное программирование на C++ или клиентское веб-программирование.

В 2001 году произошло событие, ознаменовавшее важную победу фирмы Microsoft в отрасли веб-программирования, — появилась платформа ASP.NET. Выход ASP.NET открыл путь в веб-программирование огромному количеству профессионалов и внес важный вклад в изменение модели разработки веб-приложений. Впрочем, технология ASP.NET возникла не на пустом месте. Она продолжила линию как минимум двух предшествующих технологий: классической технологии ASP (Active Server Pages) и JSP (Java Server Pages).

Итак, технология ASP.NET была успешной; но что еще важнее, она была принята практически во всех новых веб-проектах последнего десятилетия, ориентированных на платформу Microsoft . В наши дни ASP.NET единогласно считается стабильной, проверенной и высокопроизводительной платформой веб-разработки.

За это время фирма Microsoft значительно усовершенствовала ASP.NET. Сегодня в ASP.NET появились точки расширения, которых не было в первоначальном варианте. Также появилась расширенная платформа для AJAX-программирования, а встроенные элементы управления были доработаны для более точного соблюдения требований CSS и XHTML.

Долгое время термином «ASP.NET» обозначались приложения, написанные с использованием модели программирования Web Forms. Выражаясь более точно, можно сказать, что термином «ASP.NET» обозначается нижележащая платформа и среда выполнения, тогда как термином «Web Forms» обозначается способ создания страниц и приложений. Почти десять лет эти два термина считались синонимами.

Однако десять лет — долгий промежуток времени, особенно в мире программирования. Сегодня существует альтернативная платформа веб-разработки — ASP.NET MVC, она растет и быстро развивается. Остается ли ASP.NET Web Forms превосходным решением для компаний, занимающихся разработкой веб-приложений? Является ли модель Web Forms лучшей из всех возможных моделей? Стоит ли искать альтернативные методы?

Хотя эта книга посвящена архитектуре веб-приложений для платформы ASP.NET 4 с использованием модели Web Forms, ее первая глава содержит подробный обзор модели Web Forms, а также пытается дать прогноз направлений будущего развития технологий веб-разработки для платформы Microsoft.

ПРИМЕЧАНИЕ

В этой книге (а также в других моих книгах) термином «классическая технология ASP.NET» иногда обозначаются приложения ASP.NET, написанные в соответствии с моделью программирования Web Forms. Этот термин аналогичен термину «классическая технология ASP», который часто используется для различения технологии Active Server Pages и ASP.NET Web Forms.

Совершеннолетие ASP.NET Web Forms

Технология ASP.NET была создана в конце 1990-х годов как попытка использования практических решений, определенных разработчиками ASP. Многие из этих решений были встроены в новую платформу — и что еще лучше, сама платформа идеально интегрировалась с набирающей обороты моделью ускоренной разработки RAD (Rapid Application Development), которая были одним из определяющих факторов успеха Visual Basic.

В то время модель RAD рассматривалась как облегченная (а часто и более эффективная) альтернатива для объектно-ориентированного программирования (ООП). Благодаря поддержке RAD в визуальных конструкторах и редакторах практически любой программист мог быстро и легко создать прототип и протестировать приложение за считанные минуты. Методология RAD избавляла от лишних сложностей и аналитической работы, которые были необходимы в более теоретических (и более хлопотных?) методах — таких, как объектно-ориентированное проектирование и программирование. Рекламная кампания десятилетней давности по продвижению RAD строилась на утверждении: «Для быстрого создания хороших и эффективных программ не обязательны объектно-ориентированные методы и принципы»

Достоинства исходной модели

Модель ASP.NET Web Forms изначально проектировалась для того, чтобы реализовать потенциал RAD в среде Web. Таким образом, главным определяющим фактором для большинства основных характеристик и ключевых концепций ASP.NET было стремление к производительности программирования.

Модель Web Forms базируется на трех основных концепциях: *обратной передаче страниц*, *состоянии просмотра* и *серверных элементах управления*. Эти три концепции работают совместно в соответствии с моделью, изображенной на рис. 1.1.

Каждый запрос HTTP, передаваемый веб-серверу и сопоставляемый со средой выполнения ASP.NET, проходит несколько стадий, в которых центральное место

занимает обработка события *обратной передачи* (postback). Событие обратной передачи — главное действие, которое ожидает получить пользователь в результате обработки своего запроса.

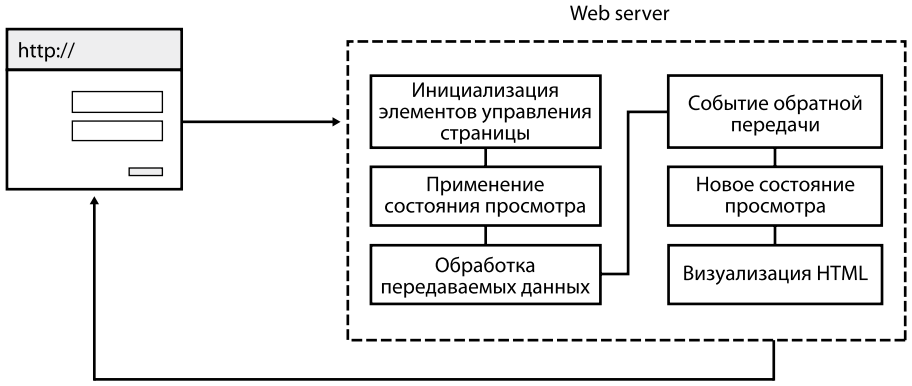

Рис. 1.1. Модель Web Forms

Сначала происходит обработка запроса для извлечения подготовительной информации, необходимой для успешного действия обратной передачи. В эту информацию входит состояние элементов управления, совместно формирующих итоговую разметку HTML-страницы. После обратной передачи для браузера генерируется ответ HTML с новым состоянием элементов, которое будет использоваться при следующем запросе.

Все операции на стороне сервера связываются воедино в соответствии с определением паттерна «Контроллер страницы». В свете этого обстоятельства каждый запрос рассматривается как обрабатываемый специальной сущностью — контроллером, который в конечном счете отвечает за вывод страницы HTML. Контроллер страницы реализуется в виде класса, инициирующего несколько событий в коде разработчика, тем самым предоставляя разработчику механизм взаимодействия с запросом и влияния на конечный результат.

Чтобы лучше понять суть модели Web Forms и причины ее успеха, рассмотрим следующий фрагмент кода:

```
void Button1_Click(Object sender, EventArgs args)
{
   Label1.Text = TextBox1.Text;
}
```

Функция `Button1_Click`, определяемая в классе Web Forms, представляет обработчик события обратной передачи. Когда пользователь щелкает на элементе HTML с соответствующим идентификатором (в данном случае `Button1`), генерируется запрос, который разрешается посредством выполнения приведенного кода. Если бы не природа веб-протоколов, не поддерживающих состояния, все выглядело бы как стандартная модель событийно-управляемого программирования, которым многие из нас пользовались (и даже с удовольствием) на заре эпохи Visual Basic в конце 1990-х годов.

В теле метода обработчика можно напрямую обращаться к любым другим элементам страницы и задавать их состояние, как при прямом выполнении операций с элементами пользовательского интерфейса.

Однако следует заметить, что приведенный выше код выполняется на веб-сервере, поэтому потребуется дополнительная работа для согласования клиентского кода HTML с серверной средой. Но этот механизм работает, и он прост — чрезвычайно прост! — для понимания и применения.

Обратная передача страницы

Страница ASP.NET базируется на одном компоненте формы, содержащем все элементы ввода, с которыми может взаимодействовать пользователь. Форма также содержит элементы отправки данных — например, кнопки или ссылки.

При отправке данных формой содержимое текущей формы отправляется на серверный URL-адрес — по умолчанию совпадающий с URL-адресом текущей страницы. Действие по отправке содержимого той же странице называется *обратной передачей* (postback). В ASP.NET страница отправляет все содержимое своей уникальной формы самой себе. Иначе говоря, страница представляет собой структурный блок приложения, содержащий как визуальный интерфейс, так и логику обработки действий пользователя.

Щелчок на кнопке или ссылке отправки данных приказывает браузеру запросить у веб-сервера новый экземпляр той же страницы. При этом браузер также отправляет все содержимое, доступное на (единственной) форме страницы. На сервере исполнительное ядро ASP.NET обрабатывает запрос и в конечном итоге выполняет некоторый код. Следующий пример демонстрирует связь между компонентом кнопки и выполняемым кодом обработчика:

```
<asp:Button runat="server" ID="Button1" OnClick="Button1_Click" />
```

Выполняемый код представляет собой серверный обработчик для исходного события на стороне клиента. В коде обработчика программист может обновить пользовательский интерфейс, изменяя состояние серверных элементов управления, как уже было показано ранее:

```
public void Button1_Click(object sender, EventArgs args)
{
    // В элементе Label выводится содержимое текстового поля
    Label1.Text = "The textbox contains: " + TextBox1.Text;
}
```

Во время выполнения кода обработчика все серверные элементы на странице уже были обновлены, а их состояние точно соответствует состоянию на момент последнего запроса к странице, с учетом всех изменений, обусловленных отправкой данных. В настольных приложениях подобное поведение с отслеживанием состояния считается само собой разумеющимся, но в ASP.NET оно требует фокусов с обратной передачей.

Состояние просмотра

Состояние просмотра (view state) представляет собой словарь, используемый страницами ASP.NET для хранения состояния своих дочерних элементов управления между обратными передачами. Состояние просмотра играет ключевую роль в реализации модели обратной передачи. Без него отслеживание состояния в ASP.NET было бы невозможно.

До выхода ASP.NET — в классическом программировании на базе VBScript — разработчики часто использовали скрытые поля для отслеживания критических данных между двумя последовательными запросами. Такое решение было необхо-

димо при использовании на странице нескольких форм HTML. Отправка данных одной формой приводила к сбросу значений в полях другой. Для решения этой проблемы отслеживаемые значения хранились в скрытом поле и инициализировались на программном уровне во время генерирования страницы.

Состояние просмотра всего лишь представляет собой расширенную и усовершенствованную версию этого стандартного приема. Это уникальное (и зашифрованное) скрытое поле, в котором хранится словарь значений всех элементов управления (уникальной) формы страницы ASP.NET.

По умолчанию каждый элемент страницы сохраняет свое полное состояние — включая значения всех его свойств — в состоянии просмотра. В странице среднего размера состояние просмотра занимает несколько десятков килобайт дополнительных данных. Эти данные передаются клиенту и принимаются сервером с каждым запросом страницы. Тем не менее они никогда не используются (и не должны использоваться!) на стороне клиента. За прошедшие годы размер состояния просмотра значительно сократился, но и в наши дни состояние просмотра продолжает рассматриваться как серьезная дополнительная нагрузка для канала связи.

Конечно, можно написать страницу, сводящую к минимуму использование состояния просмотра для ускорения загрузки, однако состояние просмотра все равно остается фундаментальным компонентом архитектуры ASP.NET Web Forms. Исключение состояния просмотра из ASP.NET потребует значительной переработки всей платформы.

В ASP.NET 4 появились новые возможности, которые позволяют разработчику лучше управлять размером состояния просмотра без ущерба для функциональности страницы.

Серверные элементы управления

Серверные элементы управления играют ключевую роль в модели ASP.NET Web Forms. Вывод страницы ASP.NET определяется в виде комбинации литералов HTML и разметки серверных элементов управления ASP.NET. Серверный элемент управления представляет собой компонент с открытым интерфейсом, который может настраиваться с использованием тегов разметки, дочерних тегов и атрибутов. Каждый серверный элемент управления имеет уникальный идентификатор, который однозначно определяет его.

В разметке страницы ASP.NET серверные элементы управления отличаются от простых строковых литералов HTML по наличию атрибута `runat`. Все теги, не имеющие атрибута `runat`, интерпретируются как литералы HTML и передаются в выходной поток ответа без дополнительной обработки. Все, что имеет пометку `runat`, идентифицируется как серверный элемент управления.

Серверные элементы управления изолируют пользователя от фактического генерирования кода HTML и JavaScript. Программирование серверного элемента управления сводится к заданию свойств компонента, предназначенного для многократного использования. Однако при обработке серверный элемент управления генерирует код HTML. В конечном итоге программирование серверных элементов управления может рассматриваться как способ получения разметки HTML, не требующий особых знаний о ее специфическом синтаксисе и функциональных возможностях.

Серверные элементы управления потребляют информацию состояния просмотра и реализуют события обратной передачи. Кроме того, они отвечают за генериро-

вание разметки, причем делают это, не требуя значительных познаний в области HTML от разработчика.

Недостатки исходной модели

На первых порах существования ASP.NET Web Forms ограниченность контакта с HTML и JavaScript была безусловным «плюсом». Однако стремительный рост популярности AJAX в середине прошлого десятилетия изменил точку зрения на возможности веб-приложений, и что еще важнее, значительно изменил ожидания пользователей. В результате от веб-приложений требуется значительно более высокая интерактивность и скорость реагирования на действия пользователя.

Один из способов ускорения реагирования — увеличение объема сценарного кода, который выполняется в браузере только при отображении конкретной страницы. Из-за этого простого факта разработчикам потребовалась более высокая степень контроля за генерируемой разметкой.

Контроль за генерируемой разметкой HTML

Для программирования функциональности AJAX разработчик должен сделать четкие и надежные предположения о структуре модели DOM (Document Object Model), отображаемой в браузере. Оказалось, что маленькие «черные ящики» (как изначально задумывались серверные элементы управления ASP.NET) уже не являются идеальными инструментами для создания веб-страниц.

Разработчик должен быть уверен в структуре выводимого кода HTML; кроме того, он должен иметь возможность контролировать идентификаторы некоторых внутренних элементов, вставляемых в итоговую модель DOM. Принятие веб-модели в значительной области промышленного и открытого сектора привело к созданию приложений, потенциальный круг пользователей которых насчитывает миллионы людей — не обязательно обладающих достаточным опытом, иногда имеющих физические недостатки и не всегда установивших новейшую версию браузера. И при этом разработчики должны сделать так, чтобы вся эта разнородная аудитория имела доступ к единому интерфейсу и в полной мере пользовалась возможностями приложения.

Как видите, пришествие AJAX произвело настоящий переворот в одной из ключевых областей ASP.NET. Технология ASP.NET, которая изначально проектировалась для обеспечения независимости от HTML, теперь должна обеспечивать модель программирования, при которой разработчик полностью контролирует HTML. Как вы увидите в этой книге, хотя эта задача не является невозможной, она требует значительно большего внимания к настройке элементов и проектированию страниц. Кроме того, разработчик должен намного лучше разбираться в возможностях платформы.

Разделение обработки и визуализации

Технология ASP.NET сильно упростила веб-программирование и повысила производительность каждого разработчика. Для этого проектирование ASP.NET ориентировалось на пользовательский интерфейс. От разработчика страницы требовалось лишь создать страницу и написать код, работающий «за кулисами».

Страница получает ввод; страница осуществляет обратную отправку; страница определяет вывод для браузера. В контексте модели, заложенной в основу ASP.NET, любые запросы рассматриваются просто как механизм генерирования HTML

через страницу. Страница затмевает все остальное; разработчик не видит какого-либо соответствия между запросом и последующим действием сервера. Все, что он видит — это входной запрос HTTP и серверный объект страницы, который обрабатывает его и возвращает HTML.

В этой модели не существует четкого разделения между фазой обработки запроса для получения низкоуровневых данных, внедряемых в ответ (например, списка записей, которые должны отображаться в таблице), и фазой форматирования низкоуровневых данных до привлекательного, элегантного макета.

И снова вы увидите, что обеспечение разделения между обработкой и визуализацией безусловно возможно и в ASP.NET Web Forms, но вам придется проявить куда больше внимания и дисциплины при написании страниц и программной логики. На рис. 1.2 приведен расширенный вариант диаграммы на рис. 1.1 с более детализированным представлением страничного паттерна, используемого для обработки запросов в ASP.NET Web Forms. (Вскоре мы вернемся к паттерну «Контроллер страницы».)

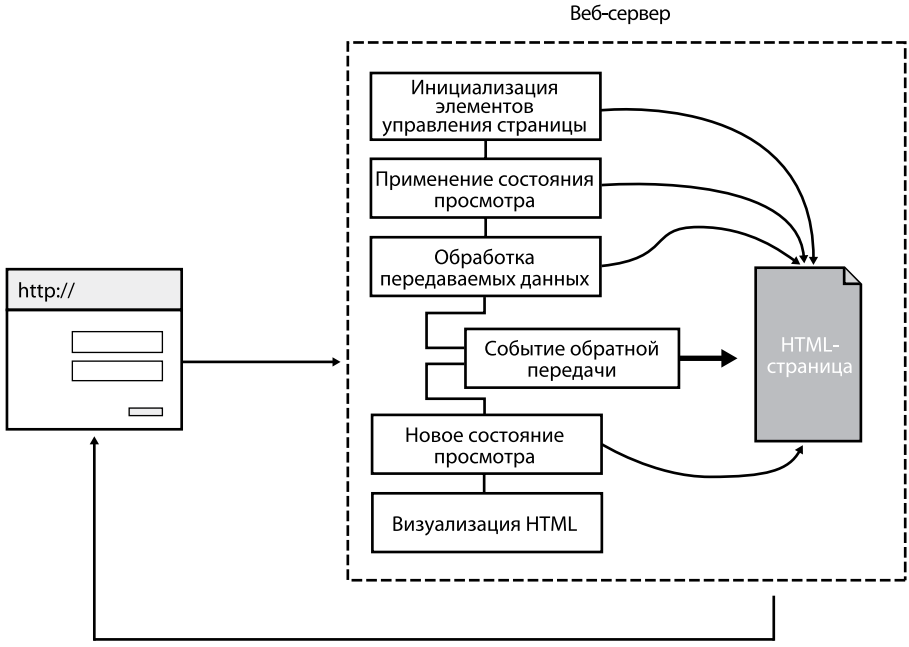

Рис. 1.2. Обработка запросов и визуализация HTML в ASP.NET

Вся обработка запроса HTTP осуществляется посредством последовательного обновления состояния серверных элементов управления, из которых состоит страница. В конце цикла текущее состояние элементов выводится в выходной поток ответа и передается браузеру. Весь цикл базируется на идее построения страницы, а не на идее выполнения действия и отображения его результатов.

Несколько лет этот аспект Web Forms принимался таким, как он есть — народ особо не жаловался, а иногда даже хвалил. Сегодня из-за растущей сложности бизнес-логики приложений, построенных на базе платформы ASP.NET, возникает необходимость в модульных тестах и статическом анализе, которые трудно организовать в среде выполнения, слишком сильно ориентированной на пользовательский интерфейс.

И снова то, что изначально казалось огромным достоинством, медленно превращается в значительный недостаток.

Облегченные страницы

Состояние просмотра — ключевой элемент модели ASP.NET, потому что оно позволяет имитировать отслеживание состояние в модели Web Forms. Многие разработчики, недавно перешедшие на ASP.NET MVC (альтернативную технологию разработки ASP.NET, полностью интегрированную в Visual Studio 2010) с трудом привыкают к тому, что каждое представление может иметь общие данные, которые приходится заполнять повторно, хотя в ходе обработки запроса вроде бы не произошло ничего, что могло бы эти данные изменить. А проще говоря, именно из-за отсутствия состояния просмотра все элементы пользовательского интерфейса (сетки данных, раскрывающиеся списки, текстовые поля) останутся пустыми, если не заполнять их явно при каждом запросе.

Состояние просмотра в ASP.NET всегда воспринималось неоднозначно. Однако, начиная с выхода ASP.NET 2.0 (около 5-ти лет назад), фирма Microsoft внесла важные изменения во внутреннюю реализацию состояния просмотра и сократила средний размер скрытого поля состояния просмотра на 40% — довольно заметное достижение.

Состояние просмотра актуально только для модели приложения, в значительной степени базирующейся на серверных элементах управления и широко использующей их для генерирования HTML. В наше время, когда проектировщики ставят под сомнение применимость классической модели ASP.NET для своих приложений и стремятся к большей интерактивности на стороне клиента, разделению ответственности (SoC, Separation of Concercs) и контролю за разметкой, состояние просмотра — один из краеугольных камней ASP.NET — уже не играет такой важной роли, как прежде. Соответственно, оно более, чем когда-либо, воспринимается как бесполезный балласт, от которого нужно избавиться.

ВНИМАНИЕ

Все больше приложений требует использования страниц с клиентским кодом, ограничивающим число обратных передач и заменяющим многие обратные передачи вызовами AJAX. В этом контексте технология Web Forms может быть адаптирована (и возможно, даже в значительной степени), но у такого подхода есть свои архитектурные ограничения, которые необходимо знать и принимать во внимание. Эти ограничения не настолько серьезны, чтобы заставить вас перейти на альтернативные технологии (например, ASP.NET MVC), но хороший проектировщик не станет относиться к ним как к пустякам, на которые можно попросту не обращать внимания.

Что делает ASP.NET, а что делаете вы?

За прошедшие десять лет технология ASP.NET Web Forms развивалась и совершенствовалась. Ее гибкая архитектура позволила внести многочисленные изменения и усовершенствования, при этом она по-прежнему остается эффективной и производительной. Хотя при проектировании архитектуры ASP.NET использовались совсем не те принципы и приоритеты, которые применяются сегодня, большинство уже упоминавшихся недостатков ASP.NET (тяжеловесные страницы, ограниченный контроль над разметкой, трудности с тестированием) удается сгладить или устранить для построения эффективных решений. Таким образом, появление новых технологий вроде ASP.NET MVC не означает, что технология ASP.NET Web Forms (и вместе

с ней ваши навыки) осталась в прошлом. Всегда существуют веские причины для разработки новых технологий, но хорошее понимание потребностей и возможностей по-прежнему остается единственным надежным путем решения проблем.

При проектировании ASP.NET Web Forms за образец был взят паттерн «Контроллер страницы». Давайте поближе познакомимся с этим паттерном и посмотрим, что можно сделать для преодоления некоторых из его недостатков.

Паттерн «Контроллер страницы»

Модель ASP.NET Web Forms передает входящий запрос компоненту обработчика HTTP (то есть попросту классу, реализующему интерфейс `IHttpHandler`). В соответствии с моделью ASP.NET Web Forms, обработчик HTTP должен вернуть код HTML для браузера. (обработчики HTTP более подробно описаны в главе 4). Способ подготовки кода HTML для браузера в значительной мере ориентирован на создание веб-страницы, а в его основу заложен паттерн «Контроллер страницы».

Этот паттерн интерпретирует обработку запроса как задачу, которая проходит через ряд фаз: создание экземпляра страницы, инициализация страницы, восстановление состояния страницы, обновление страницы, визуализация страницы и выгрузка страницы. Некоторые из этих фаз были представлены на рис. 1.2, и все они подробно обсуждаются в главах 2 и 3.

Реализации паттерна начинается с базового класса страницы и определяет стратегию обработки запроса — жизненный цикл страницы. Реализация жизненного цикла страницы формирует интерфейс виртуальных методов и событий, которые должны переопределяться и обрабатываться производными страницами (рис. 1.3).

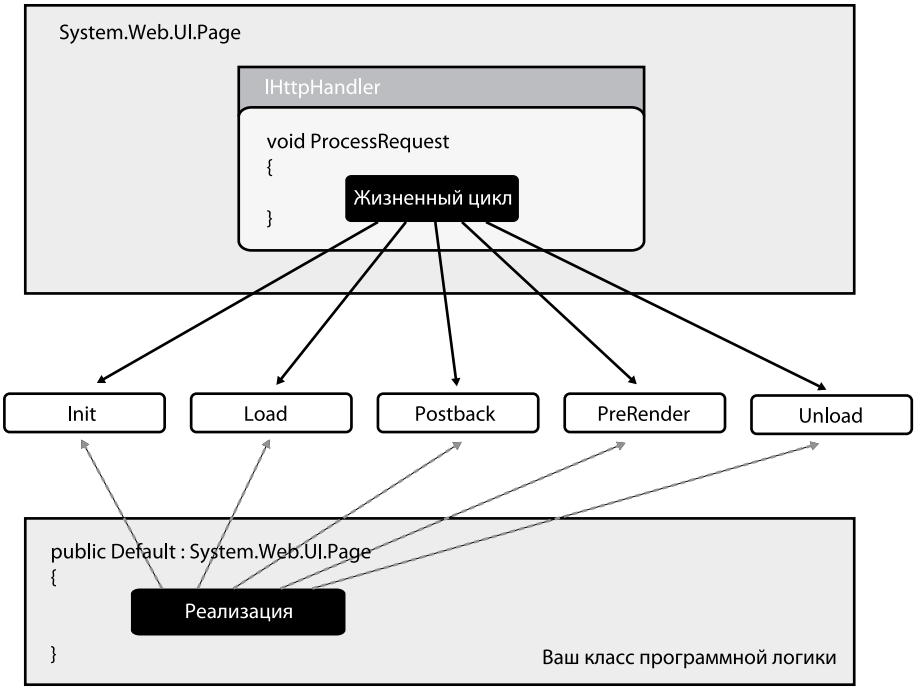

Рис. 1.3. Внутренний жизненный цикл страницы, доступный для пользовательского кода через классы контроллеров страниц

Производные классы страниц в терминологии ASP.NET называются *классами программной логики* (code-behind classes). Написание страницы ASP.NET в конечном итоге означает написание класса программной логики с добавлением предполагаемого пользовательского интерфейса. В классе программной логики хранится вся логика, необходимая для обслуживания входящих запросов, источником которых являются элементы ввода на странице. Класс программной логики является производным от системного класса `System.Web.UI.Page`.

Сам по себе класс программной логики представляет собой объект-«контроллер», отвечающий за обработку конкретного запроса. С другой стороны, в контексте приложения это может привести к формированию небольшой иерархии классов, изображенной на рис. 1.4.

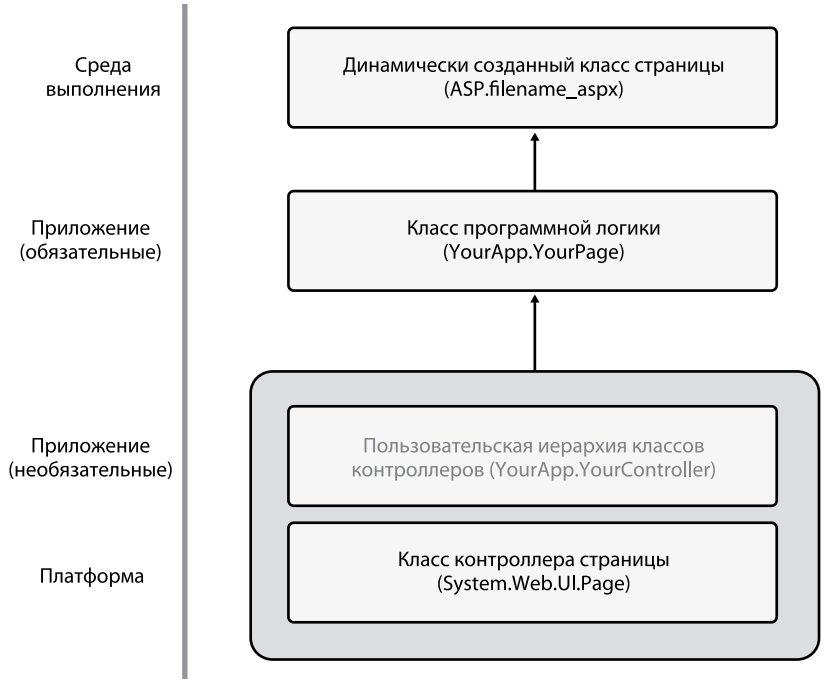

Рис. 1.4. Пример иерархии классов

Ваш класс программной логики может быть производным от системного базового класса или от одного из промежуточных классов, обладающих расширенным поведением. Разработчики могут расширять иерархию, изображенную на рисунке, по своему усмотрению. Особенно в крупных приложениях может быть полезно создать промежуточные классы страниц для моделирования сложных представлений и реализации сложных правил навигации. По сути, построение пользовательской иерархии классов страниц означает размещение пользовательских классов между контроллером страницы и фактическим классом программной логики.

Главной причиной для создания пользовательских иерархий страниц является настройка контроллера страницы с целью предоставления разработчикам доступа к адаптированному жизненному циклу. Промежуточный класс обычно включает блоки общего поведения и предоставляет в распоряжение разработчиков новые события и переопределяемые методы.