Оглавление

Содержание

Вступительное слово от сообщества

.NET уже много лет является одним из лидирующих фреймворков для разработки веб-приложений. Пройдя длинный путь от ASP.NET до современного ASP.NET Core, он вобрал в себя все лучшие подходы к разработке приложений с отрисовкой на стороне сервера и веб-API-приложений. ASP.NET Core - продукт с открытым исходным кодом, каждый может изучить любой аспект его работы. Однако объём кода велик, и не так-то просто сразу понять, что искать и как разбираться с ним. Microsoft предоставляет отличную документацию по основам серверной веб-разработки и ASP.NET Core на официальном сайте, однако этого может быть недостаточно для выстраивания целостной картины.

Именно поэтому данная книга очень ценна. Автор превосходно знает ASP.NET Core, работал с ним с первых версий и как никто другой понимает, какие аспекты фреймворка наиболее важны для его успешного использования. Разработчику, помимо работы с основной логикой приложения, важно понимать, как работать с настройками, журналированием, авторизацией, как обеспечивать безопасность приложений. Все эти темы тщательно рассмотрены в книге. Автору удалось охватить широту фреймворка, рассмотрев большое количество различных аспектов, и при этом достаточно глубоко разобрать многие из них. Все это позволяет рассматривать эту книгу как отличный способ подробного знакомства с разработкой серверных приложений на .NET.

Систематизированной информации об ASP.NET Core на русском языке мало. Фреймворк быстро развивается, постоянно появляются новые термины, и даже те, что давно используются, не всегда имеют устоявшийся перевод. Мы обсуждали, спорили, думали о том, как читатели будут искать термины в сети интернет, как они звучат в неформальных беседах. Что-то получилось хорошо, что-то не очень, но в целом мы довольны результатом и рады, что такая интересная и полезная книга есть теперь и на русском языке. Отдельная благодарность автору за простые и по-

нятные примеры кода и отличные иллюстрации, наглядно демонстрирующие объясняемые концепции.

Добро пожаловать в мир ASP.NET Core, и приятного чтения!

Российское сообщество .NET разработчиков DotNet.Ru

Над переводом работали представители сообщества DotNet.Ru:

- Игорь Лабутин;
- Андрей Беленцов;
- Максим Шошин;
- Вадим Мингажев;
- Сергей Бензенко;
- Радмир Тагиров;
- Эмиль Янгиров;
- Анатолий Кулаков.

Предисловие

ASP.NET Core 5.0 появился в 2020 году, более чем через четыре года после выпуска ASP.NET Core 1.0, но ASP.NET также имеет долгую историю, которая послужила основой и стимулом для развития ASP.NET Core.

Microsoft выпустила первую версию ASP.NET в 2002 году как часть платформы .NET Framework 1.0. С тех пор она прошла несколько выпусков, в каждом из которых были добавлены функции и расширяемость. Однако каждый выпуск был построен на основе .NET Framework, поэтому она предустановлена во всех версиях Windows.

Это приносит смешанные преимущества: с одной стороны, сегодня ASP.NET 4.x является надежной, проверенной в боях платформой для создания современных приложений для ОС Windows. С другой стороны, она ограничена этой зависимостью – изменения в базовой платформе .NET Framework имеют далеко идущие последствия, в результате чего наблюдается замедление скорости развертывания, а это оставляет за бортом многих разработчиков, создающих и развертывающих приложения для Linux или macOS.

Когда я впервые начал изучать ASP.NET Core, я был одним из таких разработчиков. Будучи в душе пользователем Windows, я получил от своего работодателя компьютер с macOS и поэтому все время работал на виртуальной машине. ASP.NET Core обещал все это изменить, позволив вести разработку и на компьютере с Windows, и на компьютере с macOS.

Можно сказать, что я опоздал во многих отношениях, проявляя активный интерес только перед выходом релиза-кандидата ASP.NET Core RC2. К тому моменту существовало уже восемь бета-версий, многие из которых содержали существенные критические изменения. Не погружаясь во все это полностью до выхода RC2, я избавился от сырых инструментов и меняющихся API.

То, что я увидел в тот момент, меня очень впечатлило. ASP.NET Core позволяет разработчикам использовать имеющиеся у них знания о платформе .NET и приложениях ASP.NET MVC, в частности используя текущие передовые практики, такие как внедрение зависимостей, строго типизированная конфигурация и журналирование. Кроме того, мож-

но было создавать и развертывать кросс-платформенные приложения. Я не устоял.

Эта книга появилась во многом благодаря моему подходу к изучению ASP.NET Core. Вместо того чтобы просто читать документацию и статьи в блогах, я решил попробовать что-то новое и начать писать о том, что я узнал. Каждую неделю я посвящал время изучению нового аспекта ASP.NET Core и писал об этом сообщение в блоге. Когда появилась возможность написать книгу, я ухватился за этот шанс – это еще один повод подробно изучить фреймворк!

С тех пор, как я начал писать эту книгу, многое изменилось как в отношении самой книги, так и ASP.NET Core. Первый крупный выпуск фреймворка в июне 2016 года по-прежнему имел много шероховатостей, в частности что касалось работы с инструментами. С выпуском .NET 5.0 в ноябре 2020 года ASP.NET Core действительно стал самостоятельным: API и инструменты достигли зрелого уровня. Данная книга нацелена на версию .NET 5.0 для ASP.NET Core, но если вы используете хотя бы версию .NET Core 3.1, то сможете без проблем работать с этим изданием.

В этой книге рассказывается обо всем, что вам нужно для начала работы с ASP.NET Core, независимо от того, новичок ли вы в веб-разработке или уже являетесь разработчиком ASP.NET. В ней очень много внимания уделяется самому фреймворку, поэтому я не буду вдаваться в подробности, касающиеся клиентских фреймворков, таких как Angular и React, или таких технологий, как Docker. Я также не описываю все новые функции .NET 5.0, такие как Blazor и gRPC. Вместо этого я даю ссылки, по которым вы можете найти дополнительную информацию.

Мы сосредоточимся на создании приложений с отрисовкой на стороне сервера, используя страницы Razor и веб-API, применяя контроллеры MVC. Вы познакомитесь с основами ASP.NET Core, такими как промежуточное ПО, внедрение зависимостей и конфигурация, а также узнаете, как настроить их в соответствии со своими требованиями. Вы узнаете, как добавить аутентификацию и авторизацию в свои приложения, как повысить их безопасность, а также как развертывать их и осуществлять мониторинг. Наконец, вы узнаете, как тестировать приложения, используя модульные и интеграционные тесты.

Лично мне приятно работать с приложениями ASP.NET Core по сравнению с приложениями, использующими предыдущую версию ASP.NET, и надеюсь, что эта страсть проявится в данной книге!

Об этой книге

Данная книга посвящена фреймворку ASP.NET Core: в ней рассказывается о том, что это такое и как использовать его для создания веб-приложений. Хотя часть этой информации уже доступна в интернете, она разбросана по сети в виде разрозненных документов и сообщений в блогах. Эта книга показывает, как создать свое первое приложение, наращивая сложность по мере того, как вы будете закреплять предыдущие концепции.

Я представляю каждую тему на относительно небольших примерах, вместо того чтобы создавать одно-единственное приложение на протяжении всей книги. У обоих подходов есть свои достоинства, но я хотел убедиться, что основное внимание уделяется конкретным изучаемым темам, без умственных затрат на навигацию по растущему проекту.

К концу книги вы должны иметь твердое представление о том, как создавать приложения с помощью ASP.NET Core, знать сильные и слабые стороны фреймворка и как использовать его функции для безопасного создания приложений. Хотя я не трачу много времени на архитектуру приложений, я непременно привожу передовые практики, особенно там, где лишь поверхностно рассказываю об архитектуре для краткости.

Кому адресована эта книга

Данная книга рассчитана на разработчиков на языке C#, которые заинтересованы в изучении кросс-платформенного веб-фреймворка. Она не предполагает, что у вас есть какой-либо опыт создания веб-приложений, например вы можете разрабатывать приложения для мобильных устройств или ПК, хотя предыдущий опыт работы с ASP.NET или другим веб-фреймворком, несомненно, полезен.

Помимо практических знаний C# и .NET, предполагается наличие знания общих объектно-ориентированных практик и базового понимания реляционных баз данных в общем. Я предполагаю, что вы немного знакомы с HTML и CSS, а также с тем, что JavaScript является языком сценариев на стороне клиента. Вам не нужно знать JavaScript- или CSS-фреймворки

для работы с этой книгой, хотя ASP.NET Core хорошо работает с ними, если это ваша сильная сторона.

Веб-фреймворки естественным образом затрагивают широкий круг тем, начиная с базы данных и сетью и заканчивая визуальным дизайном и написанием скриптов на стороне клиента. Я предоставляю как можно больше контекста и включаю ссылки на сайты и книги, где можно получить более подробную информацию.

Как организована эта книга: дорожная карта

Эта книга состоит из трех частей, 23 глав и трех приложений. В идеале вы должны прочитать ее от корки до корки, а затем использовать ее в качестве справочника, но я понимаю, что такой вариант подойдет не всем. Хотя я использую небольшие примеры приложений для демонстрации той или иной темы, некоторые главы основаны на предыдущих, поэтому содержание книги будет иметь больше смысла, если вы будете читать главы последовательно.

Я настоятельно рекомендую читать главы первой части последовательно, поскольку каждая глава основывается на темах, представленных в предыдущих главах. Вторую часть также лучше читать последовательно, хотя большинство глав независимы, если вы хотите перескакивать от одной темы к другой. Главы в третьей части можно читать в произвольном порядке, хотя я рекомендую делать это только после того, вы прошли первую и вторую части.

Первая часть представляет собой общее введение в ASP.NET Core и дает общую архитектуру типичного веб-приложения. Изучив основы, мы переходим к фреймворку Razor Pages, который составляет основную часть веб-страниц, веб-приложений ASP.NET Core с отрисовкой на стороне сервера и базовой архитектуры *модель–представление–контроллер* (MVC):

- глава 1 знакомит вас с ASP.NET Core и его местом в среде веб-разработки. В ней обсуждается, когда следует и когда не следует использовать ASP.NET Core, основы веб-запросов в ASP.NET Core и варианты, доступные для окружения разработки;
- в главе 2 рассматриваются все компоненты базового приложения ASP.NET Core, обсуждаются их роли и то, как они сочетаются для генерации ответа на веб-запрос;
- в главе 3 описывается конвейер промежуточного ПО, который является основным конвейером приложения в ASP.NET Core. Он определяет, как обрабатываются входящие запросы и как должен генерироваться ответ;
- в главе 4 показано, как использовать Razor Pages для создания страничных веб-сайтов. Razor Pages – это рекомендуемый способ создания приложений с отрисовкой на стороне сервера в ASP.NET Core, предназначенный для страничных приложений;
- в главе 5 описана система маршрутизации Razor Pages. Маршрутизация – это процесс сопоставления URL-адресов входящих запросов

с определенным классом и методом, который затем выполняется для генерации ответа;
- в главе 6 рассматривается привязка модели – процесс сопоставления данных формы и параметров URL-адреса, передаваемых в запросе, с конкретными объектами C#;
- в главе 7 показано, как создавать HTML-страницы с помощью языка шаблонов Razor;
- глава 8 основывается на главе 7, вводя тег-хелперы, которые могут значительно сократить объем кода, необходимого для создания форм и веб-страниц;
- в главе 9 описывается, как использовать контроллеры MVC для создания API-интерфейсов, которые могут вызываться клиентскими приложениями.

Вторая часть охватывает важные темы для создания полнофункциональных веб-приложений, после того как вы разберетесь с основами:

- в главе 10 описывается, как использовать встроенный контейнер внедрения зависимостей ASP.NET Core для настройки сервисов вашего приложения;
- в главе 11 обсуждается, как считывать параметры и секреты в ASP.NET Core и как отображать их в строго типизированные объекты;
- глава 12 знакомит с библиотекой Entity Framework Core, которая используется для сохранения данных в реляционной базе данных;
- глава 13 основывается на темах первой части, знакомя вас с конвейером фильтров MVC и Razor Pages;
- в главе 14 описывается, как добавить профили пользователей и аутентификацию в свое приложение с помощью ASP.NET Core Identity;
- глава 15 основывается на предыдущей главе, знакомя вас с авторизацией для пользователей, чтобы можно было ограничить страницы, к которым может получить доступ зарегистрированный пользователь;
- в главе 16 рассматривается, как опубликовать приложение, настроить его для промышленного окружения и как оптимизировать ресурсы на стороне клиента.

Главы, составляющие третью часть, охватывают важные сквозные аспекты разработки ASP.NET Core:

- в главе 17 показано, как настроить журналирование в приложении и как писать сообщения журнала в несколько мест;
- в главе 18 исследуются соображения относительно безопасности, которые следует учитывать при разработке приложения, в том числе настройка приложения для использования протокола HTTPS;
- в главе 19 описывается, как создавать и использовать различные специальные компоненты, такие как специальное промежуточное ПО, и как обрабатывать требования к комплексной конфигурации;
- глава 20 расширяет предыдущую главу и показывает, как создавать специальные компоненты Razor Page, такие как специальные тег-хелперы и атрибуты валидации;

- в главе 21 обсуждается новый интерфейс `IHttpClientFactory` и то, как использовать его для создания экземпляров `HttpClient` для вызова удаленных API;
- в главе 22 рассматривается обобщенная абстракция `IHost`, которую можно использовать для создания служб Windows и демонов Linux. Вы также научитесь запускать задачи в фоновом режиме;
- в главе 23 показано, как протестировать приложение ASP.NET Core с помощью фреймворка тестирования xUnit. В ней рассказывается о модульных и интеграционных тестах с использованием Test Host.

В трех приложениях представлена дополнительная информация:

- в приложении A описано, как настроить окружение разработки, если вы используете Windows, Linux или macOS;
- приложение B содержит сведения о .NET 5.0, .NET Core и .NET. Standard, обсуждает, как они вписываются в среду .NET, и объясняет, что они значат для ваших приложений;
- приложение C содержит ряд ссылок, которые я считаю полезными при изучении ASP.NET Core.

Соглашения об оформлении программного кода

Данная книга содержит множество примеров исходного кода как в пронумерованных листингах, так и в обычном тексте. В обоих случаях исходный код отформатирован `шрифтом фиксированной ширины, подобным этому`, чтобы отделить его от обычного текста. Иногда также используется жирный шрифт, чтобы выделить код, который изменился по сравнению с предыдущими шагами, например когда в существующую строку кода добавляется новая функция.

Во многих случаях оригинальный исходный код был переформатирован; мы добавили разрывы строк и переработали отступы, чтобы использовать доступное пространство на страницах в книге. Кроме того, комментарии в исходном коде часто удаляются из листингов, если описание кода приводится в тексте. Многие листинги сопровождаются аннотациями к коду, выделяя важные концепции.

Исходный код предоставляется для всех глав, кроме первой. Исходный код для каждой главы можно просмотреть в моем репозитории GitHub на странице https://github.com/andrewlock/asp-dot-net-core-in-action-2e. ZIP-файл, содержащий весь исходный код, также доступен на сайте издателя: www.manning.com/books/asp-net-core-in-action-second-edition.

Все примеры кода в этой книге используют .NET 5.0 и были созданы с применением Visual Studio и Visual Studio Code. Чтобы собрать и запустить примеры, необходимо установить .NET SDK, как описано в приложении A.

Автор онлайн

Приобретая книгу «*ASP .Net Core в действии*», вы получаете бесплатный доступ на частный веб-форум издательства Manning Publications,

где сможете оставлять отзывы о книге, задавать технические вопросы и получать помощь от авторов и других пользователей. Чтобы получить доступ к форуму и зарегистрироваться на нем, откройте в браузере страницу https://livebook.manning.com/book/asp-net-core-in-actionsecond-edition/discussion. Подробнее о форумах Manning и правилах поведения можно узнать на странице https://livebook.manning.com/#!/discussion.

Издательство Manning обязуется предоставить своим читателям место встречи, где может состояться содержательный диалог между отдельными читателями и между читателями и автором. Но со стороны автора отсутствуют какие-либо обязательства уделять форуму какое-то определенное внимание – его присутствие на форуме остается добровольным (и неоплачиваемым). Мы предлагаем задавать автору стимулирующие вопросы, чтобы его интерес не угасал!

Форум и архивы предыдущих дискуссий будут оставаться доступными, пока книга продолжает издаваться.

Об исходном коде

Весь используемый в этой книге исходный код для книг издательства «ДМК Пресс» можно найти на сайте www.dmkpress.com или www.дмк.рф на странице с описанием соответствующей книги.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге, – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

Эндрю Лок окончил Кембриджский университет со степенью инженера по специальности «Разработка программного обеспечения» и получил докторскую степень в области обработки цифровых изображений. Последние 10 лет он профессионально занимается разработкой с использованием .NET, используя широкий спектр технологий, включая WinForms, ASP.NET WebForms, ASP.NET MVC и ASP.NET Webpages. Он занимается созданием и сопровождением приложений ASP.NET Core с момента выхода версии 1.0 в 2016 году. Эндрю ведет блог, https://andrewlock.net, посвященный ASP.NET. Core. Он часто упоминается командой ASP.NET в Microsoft, в блоге .NET и в еженедельных выступлениях сообщества. С 2017 года Эндрю является обладателем статуса Microsoft Valued Professional (MVP).

Часть I

Начало работы с ASP.NET Core

В наши дни веб-приложения встречаются повсюду, от социальных сетей и новостных сайтов до приложений на вашем телефоне. За кулисами почти всегда есть сервер, на котором запущено веб-приложение или API для HTTP. Ожидается, что веб-приложения будут бесконечно масштабируемыми, будут развертываться в облаке и будут высокопроизводительными. Приступить к работе, может быть, не так просто и в лучшие времена, а сделать это с такими высокими ожиданиями может оказаться еще более сложной задачей.

Хорошая новость для вас как читателей заключается в том, что ASP.NET Core был разработан с учетом этих требований. Если вам нужен простой веб-сайт, сложное веб-приложение для электронной коммерции или распределенная сеть микросервисов, вы можете использовать свои знания в области ASP.NET Core для создания компактных веб-приложений, соответствующих вашим потребностям. ASP.NET Core позволяет создавать и запускать веб-приложения в Windows, Linux или macOS. Это очень модульный фреймворк, поэтому вы используете только необходимые компоненты, сохраняя при этом свое приложение как можно более компактным и производительным.

В первой части вы пройдете весь путь от самого начала до создания первого веб-приложения и API. В главе 1 дается общий обзор ASP.NET Core, который будет особенно полезен, если вы новичок в веб-разработке в целом. Вы получите первое представление о полноценном приложе-

нии ASP.NET Core в главе 2, и мы рассмотрим каждый компонент приложения по очереди и увидим, как они работают вместе для генерации ответа.

В главе 3 подробно рассматривается конвейер промежуточного ПО, который определяет, как обрабатываются входящие веб-запросы и генерируется ответ. Мы рассмотрим несколько стандартных компонентов промежуточного ПО и увидим, как фреймворк Razor Pages вписывается в конвейер. В главах с 4 по 8 мы сосредоточимся на Razor Pages, который является основным подходом к генерации ответов в приложениях ASP.NET Core. В главах с 4 по 6 мы исследуем поведение самого фреймворка Razor Pages, маршрутизацию и привязку модели. В главах 7 и 8 мы рассмотрим, как создать пользовательский интерфейс для своего приложения, используя синтаксис Razor и тег-хелперы, чтобы пользователи могли перемещаться по вашему приложению и взаимодействовать с ним. Наконец, в главе 9 мы рассмотрим особенности ASP.NET Core, позволяющие создавать веб-API, и увидим, чем это отличается от создания приложений на основе пользовательского интерфейса.

В первой части много информации, но к ее концу вы будете на правильном пути к созданию простых приложений с помощью ASP.NET Core. Я неизбежно вынужден буду пропустить некоторые сложные аспекты конфигурации платформы, но вы должны получить хорошее понимание структуры фреймворка Razor Pages и того, как можно использовать его для создания динамических веб-приложений. В следующих частях книги мы подробнее погрузимся в ASP.NET Core, и вы узнаете, как настроить приложение и добавить дополнительные функции, такие как профили пользователей.

1 Начало работы с ASP.NET Core

В этой главе:

- что такое ASP.NET Core;
- что можно создать с помощью ASP.NET Core;
- преимущества и ограничения .NET Core и .NET 5.0;
- как работает ASP.NET Core.

Решение изучать новый фреймворк и использовать его для разработки – это серьезные инвестиции, поэтому важно заранее определить, подходит ли он вам. В этой главе рассказывается об ASP.NET Core: что это такое, как он работает и почему следует использовать его для создания своих веб-приложений.

Если вы новичок в том, что касается разработки на платформе .NET, то эта глава поможет вам составить представление о ней. Для тех, кто уже имеет опыт разработки на .NET, я даю рекомендации относительно того, настало ли время подумать о том, чтобы переключить свое внимание на .NET Core и .NET 5.0, а также о преимуществах, которые ASP.NET Core может предложить по сравнению с предыдущими версиями ASP.NET.

К концу этой главы вы должны вполне четко представлять себе, что такое .NET, какова роль .NET 5.0, и знать основные механизмы работы ASP.NET Core. Итак, не теряя времени, приступим!

1.1 Введение в ASP.NET Core

ASP.NET Core – это кросс-платформенный фреймворк с открытым исходным кодом для разработки веб-приложений, который можно использовать для быстрого создания динамических приложений с отрисовкой на стороне сервера. Его также можно применять, чтобы создавать HTTP API для мобильных приложений, одностраничных приложений для браузеров, например основанных на Angular и React, или других серверных приложений.

ASP.NET Core предоставляет структуру, вспомогательные функции и фреймворк для создания приложений, что избавляет вас от необходимости писать бóльшую часть кода самостоятельно. Затем код фреймворка ASP.NET Core вызывает «обработчиков», которые, в свою очередь, вызывают методы бизнес-логики вашего приложения, как показано на рис. 1.1. Эта бизнес-логика является ядром вашего приложения. Здесь вы можете взаимодействовать с другими сервисами, такими как базы данных или удаленные API, но обычно бизнес-логика не зависит напрямую от ASP.NET Core.

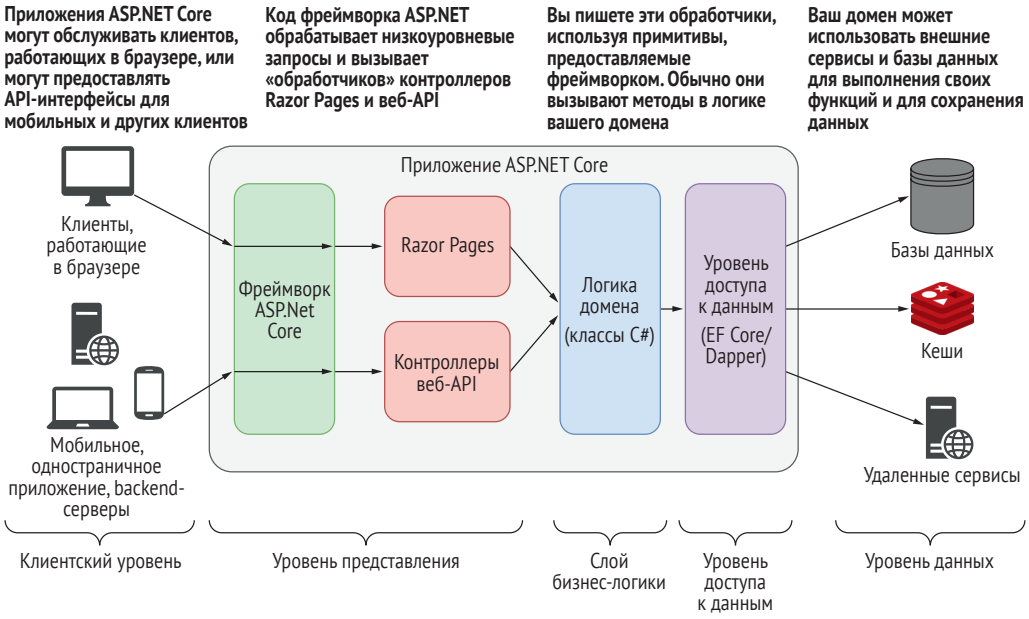

Рис. 1.1 Типичное приложение ASP.NET Core состоит из нескольких уровней. Код фреймворка ASP.NET Core обрабатывает запросы от клиента, работая со сложным сетевым кодом. Затем фреймворк вызывает обработчики (Razor Pages и контроллеры веб-API), которые вы пишете с использованием примитивов, предоставляемых фреймворком. В конце эти обработчики вызывают логику предметной области вашего приложения, которая обычно представляет собой классы и объекты C# без каких-либо зависимостей, специфичных для ASP.NET Core

В этом разделе я расскажу, для чего используется веб-фреймворк, о преимуществах и ограничениях предыдущего фреймворка ASP.NET и о том, что такое ASP.NET Core и каковы его цели.

В конце этого раздела вы должны иметь четкое представление о том, почему и для чего был создан ASP.NET Core и почему вы, возможно, захотите его использовать.

1.1.1 Использование веб-фреймворка

Если вы новичок в веб-разработке, то вам может быть непросто перейти в область, где так много модных словечек и постоянно меняющихся продуктов. Вам, наверное, интересно, неужели все это необходимо знать? Что сложного в том, чтобы вернуть файл с сервера?

Что ж, вполне возможно создать статическое веб-приложение без использования веб-фреймворка, но его возможности будут ограничены. Как только у вас появится желание обеспечить хоть какую-то безопасность или изменяемость, вы, скорее всего, столкнетесь с трудностями, и первоначальная простота, которая привлекала, исчезнет на ваших глазах.

Подобно тому, как фреймворки для разработки приложений для ПК и мобильных приложений могут помочь вам создавать нативные приложения, ASP.NET Core может ускорить и упростить написание веб-приложений, по сравнению с написанием приложения «с нуля». Он содержит библиотеки для выполнения распространенных действий, таких как:

- создание динамически изменяющихся веб-страниц;
- возможность входа пользователей в веб-приложение;
- возможность для пользователей использовать свою учетную запись Facebook для входа в ваше веб-приложение с помощью OAuth;
- обеспечение общей структуры для создания поддерживаемых приложений;
- чтение конфигурационных файлов;
- работа с изображениями;
- журналирование запросов к вашему веб-приложению.

Ключом к любому современному веб-приложению является возможность создавать динамические веб-страницы. *Динамическая веб-страница* может отображать разные данные в зависимости от вошедшего в систему пользователя или отображать контент, предоставленный пользователями. Без динамического фреймворка было бы невозможно заходить на веб-сайты или отображать какие-либо персонализированные данные на странице. Существование таких сайтов, как Amazon, eBay и Stack Overflow (см. рис. 1.2), было бы невозможным.

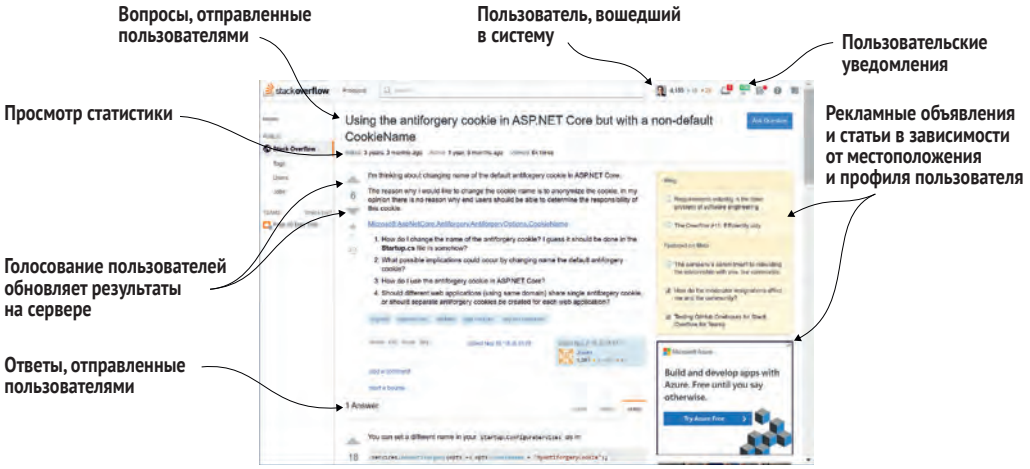

Рис. 1.2 Веб-сайт Stack Overflow (https://stackoverflow.com) создан с использованием ASP.NET и почти полностью является динамическим

Преимущества и ограничения ASP.NET

ASP.NET Core – это новейшая версия популярного веб-фреймворка ASP.NET от корпорации Microsoft, появившаяся в июне 2016 года. В предыдущих версиях ASP.NET было много поэтапных обновлений, направленных на повышение производительности разработчиков, хотя приоритет отдавался обратной совместимости. ASP.NET Core ломает этот тренд, внося значительные архитектурные изменения, ставшие результатом пересмотра способа проектирования и построения веб-фреймворка.

ASP.NET Core во многом обязан своему предшественнику ASP.NET, и многие функции, которые были перенесены в него, существовали и прежде, но ASP.NET Core – это новый фреймворк. В данном случае был переписан весь стек технологий, включая как веб-фреймворк, так и базовую платформу.

В основе изменений лежит философия, согласно которой ASP.NET должен держаться на уровне других современных фреймворков, однако разработчикам .NET по-прежнему должно быть здесь все знакомо.

Чтобы понять, *почему* корпорация Microsoft решила создать новый фреймворк, важно понимать преимущества и недостатки предыдущего веб-фреймворка ASP.NET.

Первая версия ASP.NET была выпущена в 2002 году как часть .NET Framework 1.0 в ответ на существовавшие тогда традиционные среды для написания программных сценариев – ASP и PHP. Технология Web Forms позволила разработчикам быстро создавать веб-приложения с помощью графического конструктора и простой модели событий, отражающей методы создания настольных приложений.

Фреймворк ASP.NET позволил разработчикам быстро создавать новые приложения, но со временем экосистема веб-разработки изменилась. Стало очевидно, что у технологии Web Forms много проблем, которые особенно

проявлялись при создании больших приложений. Отсутствие возможности тестирования, сложная модель с отслеживанием состояния и ограниченное влияние на сгенерированный HTML-код (что затрудняло разработку на стороне клиента) заставляли разработчиков рассматривать другие варианты.

В ответ корпорация Microsoft выпустила первую версию ASP.NET MVC в 2009 году на базе паттерна веб-разработки *модель–представление–контроллер*, широко используемого в других фреймворках, таких как Ruby on Rails, Django и Java Spring. Этот фреймворк позволил отделять элементы пользовательского интерфейса от логики приложения, упростил тестирование и предоставил более полный контроль над процессом генерирования HTML-кода.

ASP.NET MVC прошел еще четыре этапа с момента первого выпуска, но все они были построены на одном и том же базовом фреймворке, предоставляемом файлом System.Web.dll. Эта библиотека является частью .NET Framework, поэтому она предустановлена во всех версиях Windows. Она содержит весь основной код, используемый ASP.NET при создании веб-приложения.

У данной зависимости есть свои преимущества и недостатки. С одной стороны, фреймворк ASP.NET – это надежная, проверенная в боях платформа, которая отлично подходит для создания веб-приложений для Windows. Он предоставляет широкий спектр функций, которые использовались в течение многих лет, и хорошо известен практически всем веб-разработчикам, работающим с этой ОС.

С другой стороны, такая зависимость ограничивает – изменения в базовом файле System.Web.dll имеют далеко идущие последствия, что замедляет их реализацию. Это ограничивало скорость развития ASP.NET, и новые выпуски появлялись только раз в несколько лет. Также существует явная связь с веб-хостом Windows, Internet Information Service (IIS), что исключает использование фреймворка на платформах, отличных от Windows.

Совсем недавно Microsoft заявила, что с .NET Framework «завершен». Его не будут удалять или заменять, но и никакой новой функциональности он не получит. Следовательно, ASP.NET на базе System.Web.dll также не получит новую функциональность или обновления.

В последние годы многие веб-разработчики начали обращать внимание на кросс-платформенные веб-фреймворки, которые могут работать как в Windows, так и в Linux и macOS. Microsoft почувствовала, что пришло время создать фреймворк, который больше не привязан к Windows. Так на свет появился ASP.NET Core.

1.1.2 Что такое ASP.NET Core?

Разработка ASP.NET Core была обусловлена желанием создать веб-фреймворк, удовлетворяющий четырем основным требованиям:

- возможность кросс-платформенной разработки и запуск на разных аппаратных платформах и операционных системах;
- наличие модульной архитектуры для более простого обслуживания;

- фреймворк должен быть полностью разработан как программное обеспечение с открытым исходным кодом;
- соответствие текущим тенденциям в веб-разработке, таким как клиентские приложения и развертывание в облачной среде.

Для достижения всех этих целей Microsoft нужна была платформа, которая могла бы предоставить основные библиотеки для создания базовых объектов, таких как списки и словари, а также выполнять, например, простые операции с файлами. До этого момента разработка в ASP.NET всегда была привязана к платформе .NET Framework, предназначенной только для Windows и зависящей от нее. Для ASP.NET Core Microsoft создала .NET Core (а позже .NET 5.0) – легковесную платформу, работающую в Windows, Linux и macOS, как показано на рис. 1.3.

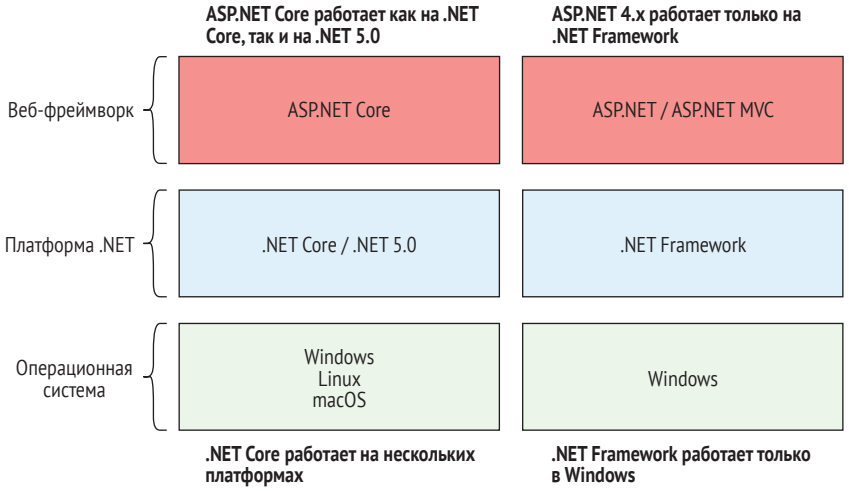

Рис. 1.3 Связь между ASP.NET Core, ASP.NET, .NET Core / .NET 5.0 и .NET Framework. ASP.NET Core работает на .NET Core и .NET 5.0, поэтому является кроссплатформенным. И наоборот, ASP.NET работает только на .NET Framework, поэтому она привязана к ОС Windows

ОПРЕДЕЛЕНИЕ .NET 5.0 – это следующая версия .NET Core, вышедшая после версии 3.1. Она представляет собой объединение .NET Core и других платформ .NET в единую среду выполнения и фреймворк. Термины .NET Core и .NET 5.0 часто используются как взаимозаменяемые, но, чтобы следовать языку Microsoft, я использую термин .NET 5.0 для обозначения последней версии .NET Core, а .NET Core – для обозначения предыдущей.

В .NET Core (и ее преемнике .NET 5.0) используется множество тех же API, что и в .NET Framework, но она более модульная и реализует только подмножество функций .NET Framework для упрощения реализации платформы и программирования для нее. Это совершенно отдельная

платформа, а не ответвление .NET Framework, хотя в ней используется аналогичный код для многих ее API.

С помощью .NET 5.0 можно создавать консольные приложения, работающие на разных платформах. Microsoft создала ASP.NET Core как дополнительный уровень поверх консольных приложений, поэтому преобразование консольного приложения в веб-приложение включает в себя добавление и организацию библиотек, как показано на рис. 1.4.

Вы пишете консольное приложение .NET 5.0, которое запускает экземпляр веб-сервера ASP.NET Core

По умолчанию Microsoft предоставляет кросс-платформенный веб-сервер под названием Kestrel

Логика вашего веб-приложения управляется Kestrel. Вы будете использовать различные библиотеки для активации таких функций, как журналирование и генерирование HTML-кода по мере необходимости

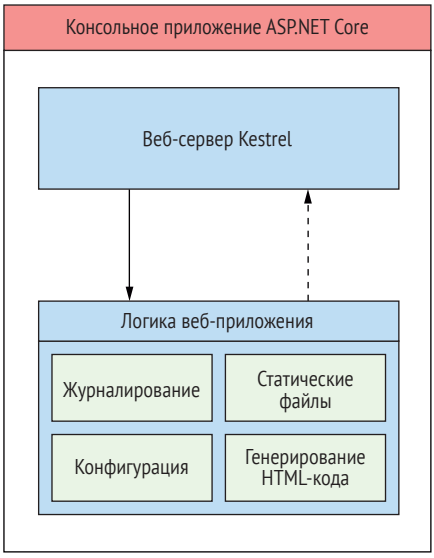

Рис. 1.4 Модель приложения ASP.NET Core. Платформа .NET 5.0 предоставляет базовую модель консольного приложения для запуска приложений командной строки. Добавляя библиотеку веб-сервера, вы преобразуете ее в веб-приложение ASP.NET Core. Дополнительные функции, такие как конфигурация и журналирование, добавляются посредством дополнительных библиотек

Добавив веб-сервер ASP.NET Core к своему приложению на .NET 5.0, вы тем самым даете ему возможность работать как веб-приложение. ASP.NET Core состоит из множества небольших библиотек. Вы можете выбирать, какую функциональность предоставить своему приложению. Вряд ли вам понадобятся все доступные библиотеки. Добавляйте только то, что нужно. Здесь есть распространенные библиотеки, которые будут присутствовать практически в каждом создаваемом вами приложении, например библиотеки для чтения конфигурационных файлов или журналирования. Другие библиотеки строятся на основе этих базовых возможностей и используются для обеспечения конкретных функций приложения, таких как вход с использованием сторонних сервисов вроде Facebook или Google.

Большинство библиотек, которые вы будете использовать в ASP.NET Core, можно найти на GitHub в репозиториях Microsoft ASP.NET Core по адресу https://github.com/dotnet/aspnetcore. Здесь вы найдете основные

библиотеки, например для аутентификации и журналирования, а также множество других внешних библиотек, например для аутентификации с участием сторонних сервисов.

Все приложения ASP.NET Core будут следовать единому шаблону в базовой версии, как предлагается стандартными библиотеками, но в целом фреймворк достаточно гибкий и позволяет создавать собственные стандарты написания кода. Эти стандартные библиотеки, библиотеки расширений, построенные на их основе, и стандарты проектирования – все это объединено под несколько расплывчатым термином ASP.NET Core.

1.2 Когда следует отдать предпочтение ASP.NET Core

Надеюсь, теперь вы имеете общее представление о том, что такое ASP.NET Core и как он был создан. Но остается вопрос: стоит ли его использовать? Корпорация Microsoft рекомендует при работе с новыми веб-проектами на .NET использовать ASP.NET Core. Но переход на новый веб-стек или его изучение – серьезный шаг для любого разработчика либо компании. В этом разделе я расскажу о том, какие приложения можно создавать с помощью ASP.NET Core, о некоторых основных характеристиках этого фреймворка, почему следует подумать об использовании ASP.NET Core при создании новых приложений, а также о том, что следует учитывать перед переносом уже существующих ASP.NET-приложений на ASP.NET Core.

1.2.1 Какие типы приложений можно создавать?

ASP.NET Core – универсальный веб-фреймворк, который можно использовать для множества приложений. Совершенно очевидно, что его можно применять для создания многофункциональных, динамических веб-сайтов, будь то сайты электронной коммерции, сайты с контентом или большие многоуровневые приложения – почти все то же самое, что и в предыдущей версии ASP.NET.

Когда платформа .NET Core только появилась, для создания таких сложных приложений было доступно всего несколько сторонних библиотек. После нескольких лет активной разработки ситуация изменилась. Многие разработчики обновили свои библиотеки для работы с ASP.NET Core, и множество других библиотек создавалось специально для ASP.NET Core. Например, система управления контентом с открытым исходным кодом Orchard[1] превратилась в Orchard Core[2] для работы в ASP.NET

[1] Исходный код проекта Orchard можно найти на странице https://github.com/OrchardCMS.

[2] Orchard Core (www.orchardcore.net). Исходный код можно найти на странице https://github.com/OrchardCMS/OrchardCore.

Core, а проект cloudscribe[1] (рис. 1.5) писался специально для ASP.NET Core с момента создания фреймворка.

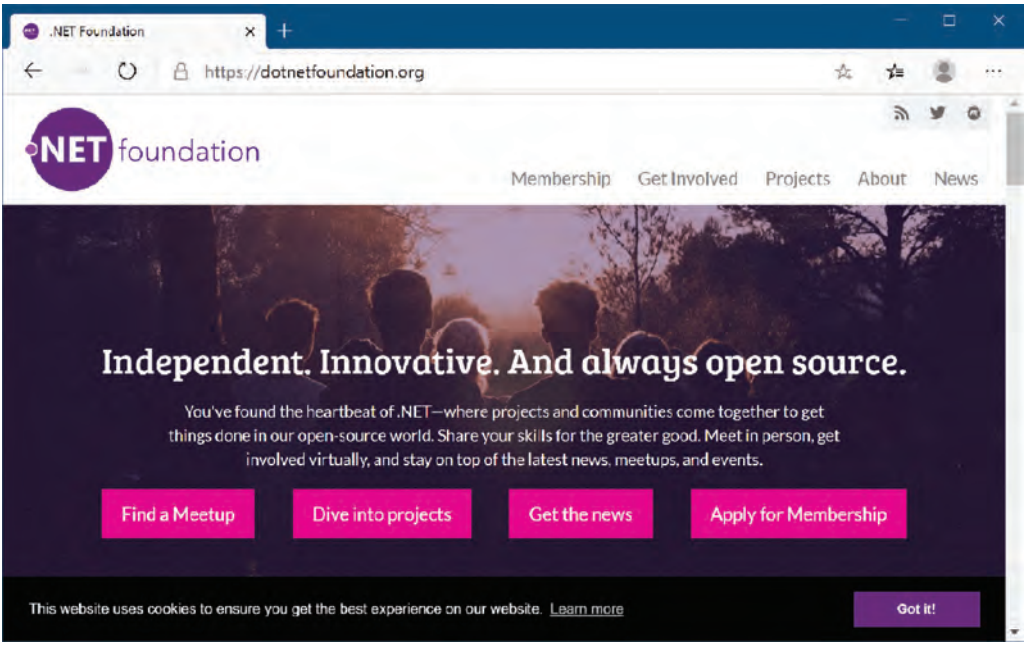

Рис. 1.5 Веб-сайт .NET Foundation (https://dotnetfoundation.org/) создан с использованием CMS cloudcribe и ASP.NET Core

Традиционные веб-приложения с отрисовкой на стороне сервера, состоящие из отдельных страниц, являются основой разработки на ASP.NET. Это касается как предыдущей версии ASP.NET, так и ASP.NET Core. Кроме того, с помощью ASP.NET Core можно легко создавать одностраничные приложения (SPA), использующие фреймворк для создания пользовательских интерфейсов, который обычно общается с REST-сервером. Независимо от того, используете ли вы Angular, Vue, React или что-либо иное, можно легко создать приложение ASP.NET Core, которое будет выступать в качестве серверного API.

ОПРЕДЕЛЕНИЕ REST – это аббревиатура, образованная от англ. Representational State Transfer — «передача состояния представления». Приложения RESTful обычно используют легковесные HTTP-вызовы без сохранения состояния для чтения, создания, обновления и удаления данных.

ASP.NET Core не ограничивается созданием RESTful-сервисов. В зависимости от требований можно легко создать веб-сервис или сервис

[1] Проект cloudcribe (www.cloudscribe.com). Исходный код можно найти на странице https://github.com/cloudscribe.

удаленного вызова процедур (RPC) для своего приложения, как показано на рис. 1.6. В простейшем случае ваше приложение может предоставлять только одну конечную точку, сужая его область действия до микросервиса. ASP.NET Core идеально подходит для создания простых сервисов благодаря тому, что является кросс-платформенным и легковесным.

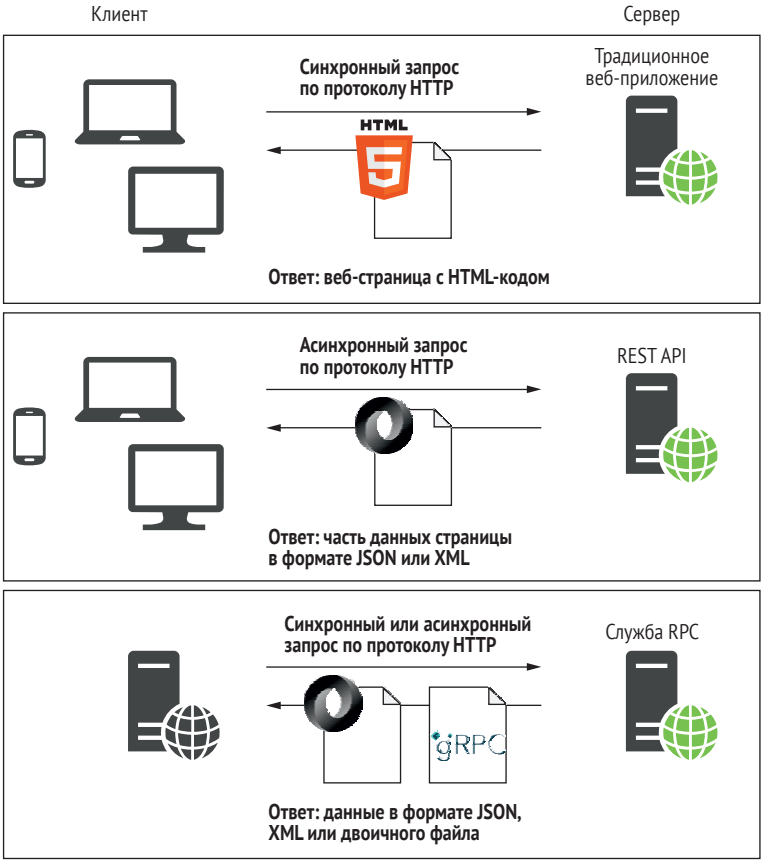

Рис. 1.6 ASP.NET Core может выступать в роли серверного приложения для различных клиентов: он может обслуживать HTML-страницы для традиционных веб-приложений, действовать как REST API для клиентских одностраничных приложений или выступать в качестве специальной службы RPC для клиентских приложений

ПРИМЕЧАНИЕ В этой книге рассказывается о создании традиционных веб-приложений с отрисовкой на стороне сервера, состоящих из отдельных страниц и RESTful веб-API. В главе 22 я также показываю, как создавать специальные приложения .NET Core, не имеющие пользовательского интерфейса – worker service.

При выборе платформы нужно учитывать несколько факторов, не все из которых являются техническими. Одним из таких факторов являет-

ся уровень поддержки, которую вы ожидаете от ее создателей. Для некоторых организаций это может быть одним из основных препятствий на пути внедрения программного обеспечения с открытым исходным кодом. К счастью, корпорация Microsoft пообещала обеспечить полную поддержку версий .NET Core и ASP.NET Core с долгосрочной поддержкой (LTS) в течение как минимум трех лет с момента их выпуска[1]. И поскольку вся разработка идет открыто, иногда можно получить ответы на свои вопросы от сообщества, а также напрямую от Microsoft.

При принятии решения об использовании ASP.NET Core необходимо учитывать два основных аспекта: являетесь ли вы уже разработчиком .NET и создаете ли вы новое приложение или хотите перенести существующее.

1.2.2 Если вы новичок в разработке на .NET

Если вы новичок в разработке на .NET и рассматриваете возможность использования ASP.NET Core, то добро пожаловать! Microsoft продвигает ASP.NET Core как привлекательный вариант для новичков в веб-разработке, а переход на кросс-платформенность означает, что он конкурирует со многими другими фреймворками на их территории. ASP.NET Core обладает множеством преимуществ по сравнению с другими кросс-платформенными веб-фреймворками:

- это современный высокопроизводительный веб-фреймворк с открытым исходным кодом;
- он использует уже знакомые паттерны проектирования и парадигмы;
- C# – прекрасный язык (если хотите, то можете использовать VB.NET или F#);
- можно выполнять сборку и запуск на любой платформе.

ASP.NET Core – это переосмысленная версия ASP.NET, созданная с использованием современных принципов проектирования программного обеспечения на основе новой платформы .NET Core / .NET 5.0. Хотя в каком-то смысле платформа .NET Core и является новой, она широко используется уже на протяжении нескольких лет и в значительной степени основана на зрелой, стабильной и надежной платформе .NET Framework, которой уже почти два десятилетия. Можете спать спокойно, зная, что, выбрав ASP.NET Core и .NET 5.0, вы получите надежную платформу, а также полнофункциональный веб-фреймворк.

Многие из доступных на сегодня веб-фреймворков используют аналогичные хорошо зарекомендовавшие себя паттерны проектирования, и ASP.NET Core ничем от них не отстает. Например, Ruby on Rails известен тем, что использует концепцию *модель–представление–контроллер* (MVC). Node.js известен способом обработки запросов с помощью не-

[1] Ознакомьтесь с политикой поддержки на странице https://dotnet.microsoft.com/platform/support/policy/dotnet-core.

больших дискретных модулей (называемых *конвейером*), а внедрение зависимостей можно найти в самых разных фреймворках. Если эти методы вам знакомы, то вам будет легко перенести их на ASP.NET Core. Если для вас все это в новинку, то можете рассчитывать на использование передовых отраслевых методик!

ПРИМЕЧАНИЕ Вы познакомитесь с конвейером в главе 3, MVC – в главе 4 и внедрением зависимостей – в главе 10.

Основным языком разработки на .NET и, в частности, ASP.NET Core является C#. У этого языка огромное количество поклонников, и не зря! Будучи объектно-ориентированным языком из семейства C, он не будет чужд тем, кто привык к C, Java и многим другим языкам. Кроме того, он имеет множество мощных функций, таких как Language Integrated Query (LINQ), замыкания и конструкции асинхронного программирования. На GitHub можно найти репозиторий C#, а также компилятор для C# от корпорации Microsoft под кодовым названием Roslyn[1].

ПРИМЕЧАНИЕ В этой книге используется C#. Я выделю несколько новых его функций, но не буду обучать вас ему с нуля. Если вы хотите изучить C#, то рекомендую четвертое издание книги *C# in Depth*, написанной Джоном Скитом (Manning, 2019), и *Code as a Pro in C#* Джорта Роденбурга (Manning, 2021).

Одним из основных преимуществ ASP.NET Core и .NET 5.0 является возможность разработки и запуска на любой платформе. Независимо от того, используете ли вы Mac, Windows или Linux, вы можете запускать одни и те же приложения ASP.NET Core и вести разработку в разных окружениях.

Что касается пользователей Linux, то здесь поддерживается широкий спектр дистрибутивов (RHEL, Ubuntu, Debian, CentOS, Fedora и openSUSE, и это лишь некоторые из них), поэтому можете быть уверены, что выбранная вами операционная система будет жизнеспособным вариантом. ASP.NET Core работает даже в крошечном дистрибутиве Alpine для компактного развертывания в контейнерах.

Помним о контейнеризации

Традиционно веб-приложения развертывались непосредственно на сервере, а в последнее время – в виртуальной машине. Виртуальные машины позволяют устанавливать операционные системы на уровне виртуального оборудования, абстрагируясь от реального аппаратного обеспечения. У этого варианта есть несколько преимуществ по сравнению с прямой установкой, например простота обслуживания, развертывания и восстановления. К со-

[1] Репозиторий исходного кода GitHub для языка C# и .NET Compiler Platform можно найти на странице https://github.com/dotnet/roslyn.

жалению, они тяжелы как с точки зрения размера файла, так и с точки зрения использования ресурсов.

Вот тут-то и пригодятся контейнеры. Контейнеры намного легковеснее и не имеют больших накладных расходов, как виртуальные машины. Они состоят из нескольких уровней и не требуют загрузки новой операционной системы при запуске. Это означает, что они быстро запускаются и отлично подходят для быстрого развертывания. Контейнеры (и, в частности, контейнеры Docker) быстро становятся популярной платформой для сборки больших масштабируемых систем.

Контейнеры никогда не были особенно привлекательным вариантом для приложений ASP.NET, но с появлением ASP.NET Core, .NET 5.0 и Docker для Windows все изменилось. Легковесное приложение ASP.NET Core, работающее на кросс-платформенном фреймворке .NET 5.0, идеально подходит для «тонкого» развертывания контейнеров. Дополнительную информацию о параметрах развертывания можно получить в главе 16.

Помимо работы на разных платформах, одним из преимуществ .NET является возможность выполнять написание кода и компиляцию только один раз. Ваше приложение компилируется в код на промежуточном языке (IL), не зависящем от платформы. Если в целевой системе установлена среда выполнения .NET 5.0, то можно запускать скомпилированный код с любой платформы. Это означает, что можно, например, вести разработку на компьютере, где установлена ОС Mac или Windows, и выполнять развертывание *абсолютно тех же файлов* на промышленных машинах с Linux. Такая возможность наконец-то была реализована с помощью ASP.NET Core и .NET Core / .NET 5.0.

1.2.3 Если вы разработчик, создающий новое приложение

Если вы разработчик .NET, то выбор, вкладывать ли средства в изучение ASP.NET Core для новых приложений, в значительной степени зависел от времени. В ранних версиях .NET Core отсутствовали некоторые функции, что затрудняло его использование. С выпуском .NET Core 3.1 и .NET 5.0 это больше не проблема; Microsoft теперь напрямую советует, что все новые приложения .NET должны использовать .NET 5.0. Microsoft пообещала предоставить исправления ошибок и безопасности для старой платформы ASP.NET, но больше не будет предоставлять никаких обновлений функциональности. .NET Framework не будет удаляться, поэтому ваши старые приложения продолжат работать, но не следует использовать ее для новых разработок.

Основные преимущества ASP.NET Core по сравнению с предыдущим фреймворком ASP.NET:

- кросс-платформенная разработка и развертывание;
- акцент на производительность;
- упрощенная модель хостинга;

- регулярные выпуски с более коротким циклом;
- открытый исходный код;
- модульные функции.

Если вы разработчик .NET и не используете какие-либо специфичные для Windows конструкции, такие как реестр, возможность сборки приложений и развертывания их на разных платформах и операционных системах открывает двери для совершенно нового направления: воспользуйтесь преимуществами более дешевого хостинга виртуальных машин Linux в облаке, используйте контейнеры Docker для повторяемой непрерывной интеграции или пишите код на своем компьютере Mac без необходимости запускать виртуальную машину Windows. Все это становится возможным благодаря ASP.NET Core в сочетании с .NET 5.0.

.NET Core и .NET 5.0 по своей сути кросс-платформенны, но при необходимости все равно можно использовать функции для каждой конкретной платформы. Например, функции для Windows, такие как реестр или службы каталогов, можно активировать с помощью пакета совместимости, который делает эти API-интерфейсы доступными в .NET 5.0[1].

Они доступны только при запуске .NET 5.0 на Windows, а не Linux или macOS, поэтому вам необходимо позаботиться о том, чтобы такие приложения работали лишь в Windows или учитывали потенциально отсутствующие API.

Модель хостинга для предыдущего фреймворка ASP.NET была относительно сложной. Она использовала Windows-сервис IIS для предоставления веб-хоста. В кросс-платформенном окружении такие симбиотические отношения невозможны, поэтому была принята альтернативная модель хостинга, отделяющая веб-приложения от основного хоста. Эта возможность привела к разработке Kestrel: быстрого кросс-платформенного HTTP-сервера, на котором может работать ASP.NET Core.

Вместо предыдущего варианта, при котором IIS вызывает определенные точки вашего приложения, приложения ASP.NET Core представляют собой консольные приложения, которые самостоятельно размещают веб-сервер и обрабатывают запросы напрямую, как показано на рис. 1.7. Эта модель хостинга концептуально намного проще и позволяет тестировать и отлаживать свои приложения из командной строки. Хотя в данном случае не обязательно избавляться от IIS (или его аналога) в промышленном окружении, как вы увидите в разделе 1.3.

ПРИМЕЧАНИЕ При желании также можно запустить ASP.NET Core внутри IIS, как показано на рис. 1.7. Это может дать преимущество в производительности по сравнению с версией с обратным прокси-сервером. Прежде всего это касается развертывания и не меняет способ построения приложений.

---

[1] Пакет обеспечения совместимости с Windows предназначен для помощи в переносе кода с .NET Framework на .NET Core / .NET 5.0. См. https://docs.microsoft.com/dotnet/core/porting/windows-compat-pack.

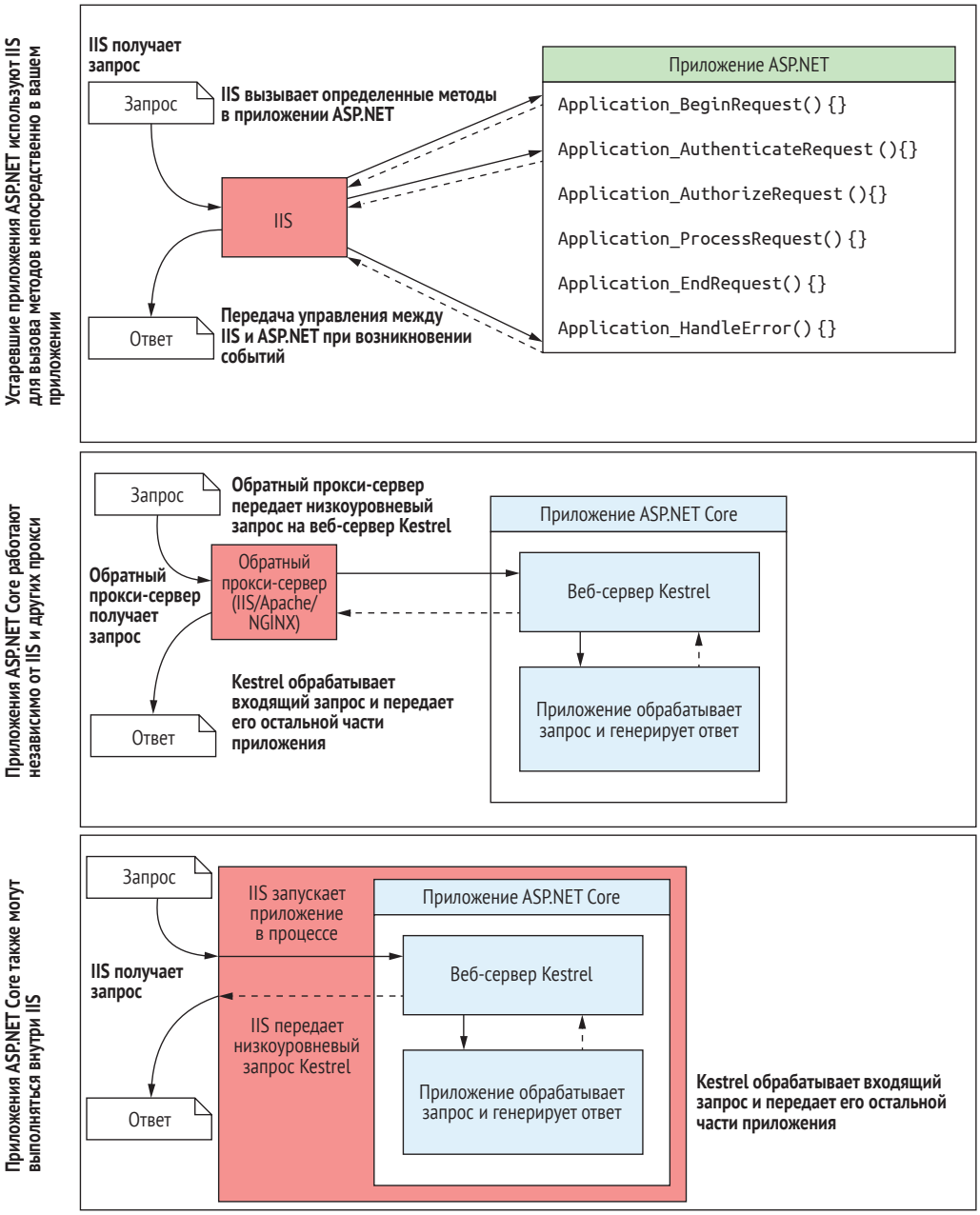

Рис. 1.7 Разница между моделями размещения в ASP.NET (вверху) и ASP.NET Core (внизу). В предыдущей версии ASP.NET IIS тесно связан с приложением. Модель размещения в ASP.NET Core проще; IIS передает запрос на локальный веб-сервер в приложении ASP.NET Core и получает ответ, но не имеет более детального представления о приложении

Изменение модели хостинга для использования встроенного веб-сервера HTTP создало еще одну возможность. В прошлом производитель-

ность была в некоторой степени проблемой для приложений ASP.NET. Конечно, можно создавать высокопроизводительные приложения – Stack Overflow (https://stackoverflow.com) является тому подтверждением, – но производительность не является приоритетом самого веб-фреймворка, поэтому в конечном итоге это может стать препятствием.

Чтобы быть конкурентоспособным среди кросс-платформенных решений, команда ASP.NET сосредоточилась на том, чтобы сделать HTTP-сервер Kestrel как можно более быстрым. TechEmpower (www.techempower.com/benchmark) уже несколько лет проводит тесты для целого ряда веб-фреймворков для разных языков. В 19-м раунде тестов в виде plaintext TechEmpower объявила, что ASP.NET Core с Kestrel стал самым быстрым из более чем 400 протестированных фреймворков[1]!

Веб-серверы: проблемы с терминологией

Один из сложных аспектов веб-программирования – это набор часто противоречащих друг другу терминов. Например, если в прошлом вы использовали IIS, то могли называть его веб-сервером или, возможно, веб-хостом. И наоборот, если вы когда-либо создавали приложение с использованием Node.js, то вы могли назвать веб-сервером само приложение. Также веб-сервером можно назвать физический компьютер, на котором работает ваше приложение!

Точно так же вы, возможно, создавали приложение для интернета и называли его веб-сайтом или веб-приложением, вероятно, несколько произвольно, исходя из наличия изменяемого контента.

В этой книге, когда я говорю «веб-сервер» в контексте ASP.NET Core, я имею в виду HTTP-сервер, который работает как часть вашего приложения ASP.NET Core. По умолчанию это веб-сервер Kestrel, но это не является обязательным требованием. При желании можно было бы написать собственный веб-сервер и использовать его вместо Kestrel.

Веб-сервер отвечает за получение HTTP-запросов и генерирование ответов. В предыдущей версии ASP.NET эту роль выполнял IIS, но в ASP.NET Core веб-сервер – это Kestrel.

В этой книге я буду использовать термин «веб-приложение» только для описания приложений ASP.NET Core, независимо от того, содержат ли они лишь статический контент или являются полностью динамическими. В любом случае это приложения, доступ к которым осуществляется по сети, поэтому этот термин кажется наиболее подходящим.

Многие улучшения производительности, внесенные в Kestrel, исходили не от самих членов команды ASP.NET, а от участников проекта с от-

[1] Как это обычно бывает в веб-разработке, технологии постоянно меняются, поэтому эти тесты со временем будут эволюционировать. Хотя ASP.NET Core может и не входить в первую десятку, вы можете быть уверены, что производительность является одним из ключевых приоритетов команды ASP.NET Core.

крытым исходным кодом на GitHub[1]. Открытая разработка означает, что обычно вы видите, что исправления и новая функциональность внедряются в промышленную эксплуатацию быстрее, чем в предыдущей версии ASP.NET, которая зависела от .NET Framework и Windows и поэтому имела длительные циклы выпуска.

Напротив, .NET 5.0, а следовательно, и ASP.NET Core предполагают регулярные небольшие изменения. Старшие версии будут выпускаться с предсказуемой периодичностью: новая версия будет выходить ежегодно, а новая версия с долгосрочной поддержкой (LTS) – каждые два года[2]. Кроме того, исправления ошибок и незначительные обновления можно выпускать по мере необходимости. Дополнительная функциональность предоставляется в виде пакетов NuGet, независимо от базовой платформы .NET 5.0.

> **ПРИМЕЧАНИЕ** NuGet – это менеджер пакетов для .NET, позволяющий импортировать библиотеки в ваши проекты. Он аналогичен Ruby Gems и npm в JavaScript или Maven в Java.

ASP.NET Core спроектирован из слабо связанных модулей, что помогает реализовать подобный подход к выпускам. Эта модульность позволяет использовать в отношении зависимостей принцип «плачу только за нужное», когда вы начинаете с «пустого» приложения и добавляете только те дополнительные библиотеки, которые вам требуются, в отличие от подхода «все сразу», использовавшегося в прежних приложениях ASP.NET. Сейчас даже необязательно применять MVC! Но не волнуйтесь, это не означает, что в ASP.NET Core мало функций; просто вам нужно подключить их. Некоторые ключевые улучшения включают в себя:

- «конвейер» с компонентами, отвечающими за обработку запроса (middleware pipeline) для определения поведения вашего приложения;
- встроенную поддержку внедрения зависимостей;
- объединенную инфраструктуру UI (MVC) и API (Web API);
- высокорасширяемую систему конфигурации;
- масштабируемость для облачных платформ по умолчанию с использованием асинхронного программирования.

Каждая из этих функций была доступна в предыдущей версии ASP.NET, но требовала значительного объема дополнительной работы для настройки. С ASP.NET Core все они уже готовы и ждут подключения!

Microsoft полностью поддерживает ASP.NET Core, поэтому, если вы хотите создать новую программную систему, нет серьезных причин не использовать его. Самое большое препятствие, с которым вы, вероятно, столкнетесь, – это желание применить модели программирования, кото-

---

[1] Проект HTTP-сервера Kestrel можно найти в репозитории ASP.NET Core на странице https://github.com/dotnet/aspnetcore.

[2] График выпуска .NET 5.0 и последующих версий: https://devblogs.microsoft.com/dotnet/introducing-net-5/.

рые больше не поддерживаются в ASP.NET Core, такие как Web Forms или сервер WCF, о чем я расскажу в следующем разделе.

Надеюсь, этот раздел пробудил в вас желание использовать ASP.NET Core для создания новых приложений. Но если вы уже работаете с ASP.NET и рассматриваете возможность переноса существующего ASP.NET приложения в ASP.NET Core, то это уже совсем другой вопрос.

1.2.4 Перенос существующего ASP.NET-приложения на ASP.NET Core

В отличие от новых приложений, существующее приложение, по-видимому, уже предоставляет ценность, поэтому переход на ASP.NET Core в конечном итоге должен принести ощутимо большую пользу, чтобы начать значительно переписывать приложение при миграции с ASP.NET на ASP.NET Core. Преимущества внедрения ASP.NET Core во многом такие же, как и у новых приложений: кросс-платформенное развертывание, модульные функции и акцент на производительность. Насколько их достаточно, во многом будет зависеть от особенностей вашего приложения, но есть характеристики, которые являются явными индикаторами *не в пользу* перехода:

- ваше приложение использует технологию Web Forms;
- ваше приложение создано с использованием WCF;
- ваше приложение большое, со множеством «продвинутых» функций MVC.

Если вы используете Web Forms, не рекомендуется переходить на ASP.NET Core. Эта технология неразрывно связана с System.Web.dll и как таковая, скорее всего, никогда не будет доступна в ASP.NET Core. Такой переход будет фактически включать в себя переписывание приложения с нуля. Это не только изменение фреймворков, но и изменение парадигм проектирования. Лучше было бы постепенно вводить концепции веб-API и пытаться уменьшить зависимость от устаревших конструкций Web Forms, таких как ViewData. В сети можно найти множество ресурсов, которые помогут вам в этом, в частности сайт https://dotnet.microsoft.com/apps/aspnet/apis[1].

Фреймворк Windows Communication Foundation (WCF) только частично поддерживается в ASP.NET Core[2]. Можно использовать какие-то сервисы WCF, но поддержка в лучшем случае будет неполноценной. Поддерживаемого способа хостинга сервисов WCF из приложения ASP.NET Core нет, поэтому если вам непременно нужно поддерживать WCF, то лучше избегать использования ASP.NET Core на данный момент.

---

[1] В качестве альтернативы можно рассмотреть возможность преобразования своего приложения в Blazor, используя усилия сообщества по созданию Blazor-версий распространенных компонентов WebForms: https://github.com/FritzAndFriends/BlazorWebFormsComponents.

[2] Клиентские библиотеки для использования WCF с .NET Core можно найти на странице https://github.com/dotnet/wcf.

СОВЕТ Если вам нравится программирование в стиле RPC, но у вас нет жестких требований к WCF, рассмотрите возможность использования gRPC. gRPC – это современный фреймворк для вызова удаленных процедур со множеством концепций, аналогичных WCF. Он поддерживается ASP.NET Core[1].

Если у вас сложное приложение и оно широко использует предыдущие точки расширения MVC, или веб-API, или обработчики сообщений, то его перенос в ASP.NET Core может оказаться более трудным. ASP.NET Core имеет множество функций, аналогичных предыдущей версии ASP.NET MVC, но базовая архитектура у него иная. Некоторые ранее существовавшие функции не имеют прямых заменителей, поэтому их придется переосмыслить.

Чем крупнее приложение, тем труднее будет перейти на ASP.NET Core. Microsoft предполагает, что перенос приложения из ASP.NET MVC в ASP.NET Core – это, по крайней мере, такого же порядка изменение, как перенос приложения из ASP.NET Web Forms в ASP.NET MVC. Если это вас не пугает, тогда все в порядке!

Если приложение используется редко и не является частью основного бизнеса или не требует значительного развития в ближайшем будущем, то я настоятельно рекомендую вам не пытаться переходить на ASP.NET Core. Microsoft будет поддерживать платформу .NET Framework в обозримом будущем (от нее зависит сама Windows!), и отдача от переноса этих «второстепенных» приложений вряд ли стоит затраченных усилий.

Итак, когда же *следует* переносить приложение на ASP.NET Core? Как я уже упоминал, наиболее подходящая возможность – это небольшие новые проекты с нуля, а не существующие приложения. Тем не менее если существующее приложение невелико или потребует значительного развития в будущем, перенос может быть хорошим вариантом. По возможности всегда лучше работать небольшими этапами, чем пытаться сделать все сразу. Но если ваше приложение состоит в основном из контроллеров MVC или веб-API и связанных представлений Razor, переход на ASP.NET Core может быть неплохим решением.

1.3 Как работает ASP.NET Core?

К настоящему времени вы должны иметь представление о том, что такое ASP.NET Core и для каких приложений следует его использовать. В этом разделе вы увидите, как работает приложение, созданное с помощью ASP.NET Core, от запроса пользователем URL до отображения страницы в браузере. Сначала мы рассмотрим, как работает HTTP-запрос к любому веб-серверу, а затем как ASP.NET Core расширяет этот процесс для создания динамических веб-страниц.

---

[1] Электронную книгу от Microsoft по gRPC для разработчиков WCF можно найти на странице https://docs.microsoft.com/en-us/dotnet/architecture/grpc-for-wcf-developers/.

1.3.1 Как работает веб-запрос по протоколу HTTP?

Как вы знаете, ASP.NET Core – это фреймворк для создания веб-приложений, которые отдают данные с сервера. Один из наиболее распространенных сценариев для веб-разработчиков – создание веб-приложения, которое можно просматривать в браузере. Процесс запроса страницы с любого веб-сервера показан на рис. 1.8.

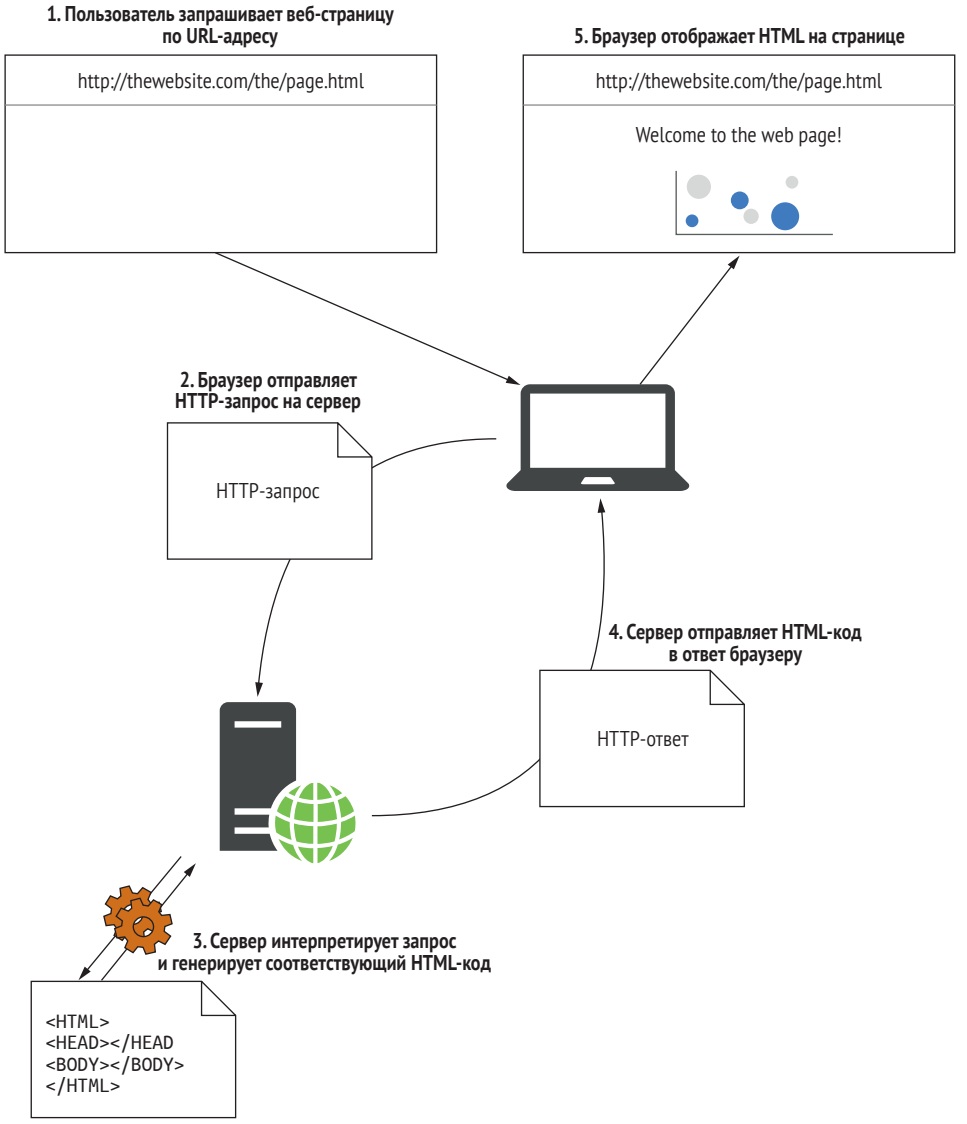

Рис. 1.8 Запрос веб-страницы. Пользователь начинает с запроса веб-страницы, что вызывает отправку HTTP-запроса на сервер. Сервер интерпретирует запрос, генерирует необходимый HTML и отправляет его обратно в HTTP-ответе. Затем браузер может отобразить веб-страницу

Процесс начинается, когда пользователь переходит на веб-сайт или вводит URL-адрес в своем браузере. URL-адрес или веб-адрес состоит из *имени хоста* и *пути* к некоему ресурсу в веб-приложении. При переходе по адресу в браузере запрос с компьютера пользователя отправляется на сервер, на котором размещено веб-приложение, с использованием протокола HTTP.

ОПРЕДЕЛЕНИЕ Имя хоста веб-сайта однозначно определяет его местоположение в интернете путем сопоставления с IP-адресом с помощью системы доменных имен (DNS). В качестве примеров можно упомянуть сайты microsoft.com, www.google.com и facebook.com.

Запрос проходит через интернет, возможно, на другую сторону света, пока наконец не попадет на сервер, связанный с данным именем хоста, на котором запущено веб-приложение. По пути запрос потенциально может быть получен и ретранслирован несколькими маршрутизаторами, но обрабатывается только после попадания на сервер, связанный с именем хоста.

Краткое руководство по HTTP

Протокол передачи гипертекста (Hypertext Transfer Protocol – HTTP) – это протокол прикладного уровня, обеспечивающий работу в сети. Это протокол, работающий по схеме «запрос–ответ» без сохранения состояния. Клиентский компьютер отправляет запрос на сервер, который, в свою очередь, передает ответ.

Каждый HTTP-запрос состоит из глагола, указывающего «тип» запроса, и пути, указывающего ресурс, с которым нужно взаимодействовать. Обычно они также содержат заголовки, представляющие собой пары «ключ–значение», а в некоторых случаях тело, например содержимое формы при отправке данных на сервер.

Ответ содержит код состояния, указывающий на то, был ли запрос успешным, а также, необязательно, заголовки и тело.

Для получения более подробной информации о самом протоколе HTTP, а также чтобы увидеть другие примеры, см. раздел 1.3 («Краткое введение в HTTP») в книге *Go Web Programming* Сау Шонга Чанга (Manning, 2016), https://livebook.manning.com/book/go-web-programming/chapter-1/point-9018-55-145-1.

Как только сервер получает запрос, он проверяет, имеет ли он смысл, и если да, то генерирует ответ. В зависимости от запроса этот ответ может быть веб-страницей, изображением, файлом JavaScript или простым подтверждением. В этом примере я предполагаю, что пользователь перешел на домашнюю страницу веб-приложения, поэтому сервер в ответ выдает некий HTML. Он добавляется к ответу, который затем отправляется обратно через интернет в браузер, отправивший запрос.

Как только браузер пользователя начинает получать ответ, он может приступить к отображению содержимого на экране, но HTML-страница может также ссылаться на другие страницы и ресурсы на сервере. Чтобы отобразить веб-страницу целиком, вместо статического бесцветного необработанного HTML-файла браузер должен повторить процесс запроса, получая каждый файл, на который есть ссылка. HTML, изображения, CSS-стили и файлы JavaScript, обеспечивающие дополнительное поведение, – все они получаются с использованием одного и того же процесса HTTP-запроса.

Практически все взаимодействия, происходящие в интернете, являются фасадом одного и того же базового процесса. Для полной отрисовки базовой веб-страницы может потребоваться всего несколько простых запросов, тогда как для современной большой веб-страницы могут потребоваться сотни. На момент написания этой главы домашняя страница Amazon.com (www.amazon.com), например, делает 606 запросов, включая запросы 3 файлов CSS, 12 файлов JavaScript и 402 файлов изображений!

Теперь, когда вы прочувствовали процесс, посмотрим, как ASP.NET Core динамически генерирует ответ на сервере.

1.3.2 Как ASP.NET Core обрабатывает запрос?

Когда вы создаете веб-приложение с помощью ASP.NET Core, браузеры по-прежнему будут использовать тот же протокол HTTP, что и раньше, для обмена данными с вашим приложением. Сам ASP.NET Core охватывает все, что происходит на сервере для обработки запроса, включая проверку действительности запроса, обработку данных для входа и генерирование HTML.

Как и в случае с примером веб-страницы, процесс запроса начинается, когда браузер пользователя отправляет HTTP-запрос на сервер, как показано на рис. 1.9.

Ваше приложение получает запрос из сети. Каждое приложение ASP.NET Core имеет встроенный веб-сервер, по умолчанию это Kestrel, который отвечает за получение необработанных запросов и создание внутреннего представления данных, объекта `HttpContext`, который может использоваться остальной частью приложения.

Из этого представления ваше приложение может получить все детали, необходимые для создания соответствующего ответа на запрос. Оно может использовать данные, хранящиеся в `HttpContext`, для формирования соответствующего ответа. Это может быть генерирование некоего HTML, возврат сообщения «Доступ запрещен» или отправка электронного письма в зависимости от требований вашего приложения.

Как только приложение завершит обработку запроса, оно вернет ответ веб-серверу. Веб-сервер ASP.NET Core преобразует представление в HTTP-ответ и отправит его в сеть, которая направит его в браузер пользователя.

Для пользователя этот процесс выглядит так же, как и при обычном HTTP-запросе, показанном на рис. 1.8: пользователь отправил запрос

и получил ответ. Все различия относятся к серверной части вашего приложения.

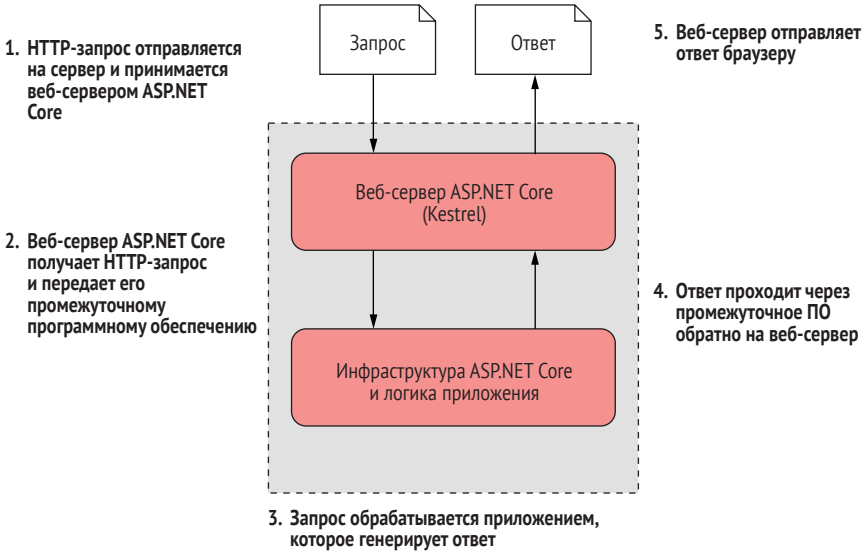

Рис. 1.9 Как приложение ASP.NET Core обрабатывает запрос. Приложение ASP.NET Core получает запрос и запускает собственный веб-сервер. Веб-сервер обрабатывает запрос и передает его телу приложения, которое генерирует ответ и возвращает его веб-серверу. Веб-сервер отправляет этот ответ браузеру

ASP.NET Core и обратные прокси-серверы

Можно предоставлять доступ к приложениям ASP.NET Core напрямую из интернета, чтобы Kestrel получал запросы непосредственно из сети. Однако чаще используется обратный прокси-сервер (reverse proxy-server) между сетью и вашим приложением. В Windows обратным прокси-сервером обычно является IIS, а в Linux или macOS это может быть NGINX, HAProxy или Apache.

Обратный прокси-сервер – это программное обеспечение, отвечающее за получение запросов и их пересылку на соответствующий веб-сервер. Обратный прокси-сервер доступен непосредственно из сети интернет, тогда как базовый веб-сервер доступен только для прокси. Такая настройка имеет несколько преимуществ, в первую очередь это безопасность и производительность веб-серверов.

Вы можете подумать, что наличие обратного прокси-сервера и веб-сервера несколько излишне. Почему бы не использовать что-то одно? Что ж, одно из преимуществ – возможность отделить ваше приложение от операционной системы сервера. Один и тот же веб-сервер ASP.NET Core, Kestrel, может быть кросс-платформенным и использоваться за различными прокси-серверами, не накладывая каких-либо ограничений на конкретную реализацию. Если вы написали новый веб-сервер, то можете использовать его вместо Kestrel, не изменяя ничего другого в своем приложении.

Еще одним преимуществом обратного прокси-сервера является то, что он может быть защищен от потенциальных угроз из общедоступного интернета. Часто прокси-серверы отвечают за дополнительные аспекты, такие как перезапуск сбойного процесса. Kestrel может оставаться простым HTTP-сервером, не беспокоясь об этих дополнительных функциях, если используется за обратным прокси-сервером. Считайте это простым разделением обязанностей: Kestrel занимается генерированием ответов по протоколу HTTP, а обратный прокси-сервер занимается обработкой подключения к интернету.

Вы видели, как запросы и ответы попадают в приложение и обратно, но я еще не коснулся того, как генерируется ответ. В первой части книги мы рассмотрим компоненты, из которых состоит типичное приложение ASP.NET Core, и то, как все они сочетаются друг с другом. В создании ответа в ASP.NET Core участвует множество компонентов, обычно на все это уходит доля секунды, но по ходу книги мы будем не спеша проходить по приложению, подробно описывая каждый из компонентов.

1.4 Что вы узнаете из этой книги

В этой книге вы подробно изучите фреймворк ASP.NET Core. Чтобы извлечь пользу из книги, вы должны быть знакомы с C# или аналогичным объектно-ориентированным языком. Также будет полезно базовое знакомство с такими веб-концепциями, как HTML и JavaScript. Вы узнаете:

- как создавать страничные приложения с помощью Razor Pages;
- ключевые концепции ASP.NET Core, такие как привязка модели (model-binding), валидация и маршрутизация;
- как генерировать HTML для веб-страниц с помощью синтаксиса Razor и функции Tag Helpers;
- как использовать такие функции, как внедрение зависимостей, конфигурирование и журналирование, по мере того как ваши приложения будут становиться более сложными;
- как защитить свое приложение, используя лучшие практики обеспечения безопасности.

На протяжении всей книги мы будем использовать множество примеров, чтобы изучать и исследовать различные концепции. Приведенные здесь примеры, как правило, небольшие и автономные, поэтому мы можем сосредоточиться на одной функции за раз.

Для большинства примеров в этой книге я буду использовать Visual Studio, но вы можете использовать свой любимый редактор или интегрированную среду разработки. В приложении A содержится подробная информация о настройке вашего редактора или интегрированной среды разработки и установке набора средств разработки .NET 5.0. Несмотря на то что в примерах этой книги показаны инструменты для Windows, все, что вы видите, в одинаковой мере применимо для платформ Linux или Mac.

СОВЕТ .NET 5.0 можно скачать на странице https://dotnet.microsoft.com/download.

В приложении А содержатся дополнительные сведения о том, как настроить среду разработки для работы с ASP.NET Core и .NET 5.0.

В следующей главе вы создадите свое первое приложение на основе шаблона и запустите его. Мы рассмотрим все основные компоненты, из которых состоит ваше приложение, и увидим, как они все вместе работают для отрисовки веб-страницы.

Резюме

- ASP.NET Core – это новый фреймворк для разработки веб-приложений, в основе которого лежат современные практики архитектуры программного обеспечения и модульность.
- Лучше всего использовать его для создания новых проектов с нуля.
- Устаревшие технологии, такие как сервер WCF и Web Forms, нельзя использовать с ASP.NET Core.
- ASP.NET Core работает на кросс-платформенной платформе .NET 5.0. Доступ к специфическим функциям Windows, таким как реестр, можно получить с помощью пакета обеспечения совместимости Windows Compatibility Pack.
- .NET 5.0 – это следующая версия .NET Core после .NET Core 3.1.
- Получение веб-страницы включает в себя отправку запроса и получение ответа по протоколу HTTP.
- ASP.NET Core позволяет динамически создавать ответы на заданный запрос.
- Приложение ASP.NET Core содержит веб-сервер, который служит точкой входа для запроса.
- Приложения ASP.NET Core обычно защищены обратным прокси-сервером, который перенаправляет запросы в приложение.