ЧАСТЬ IX

.NET Core

В этой части

ГЛАВА 31

Философия .NET Core

27 июня 2016 года в Microsoft анонсировали выход .NET Core версии 1.0 — кардинально новой платформы для всех разработчиков .NET. Новая платформа была выпущена одновременно для операционных систем (ОС) Windows, macOS и Linux. Она основывалась на языке C# и платформе .NET Framework, которую вы изучали на протяжении предшествующих тридцати глав. Начальный выпуск включал исполняющую среду .NET Core, ASP.NET Core и Entity Framework Core. С тех пор появились два дополнительных выпуска: 1.1 и 2.0 (текущая версия).

Вдобавок к возможностям межплатформенного развертывания было внесено еще одно значительное изменение. Платформа .NET Core и связанные инфраструктуры являются системами с открытым кодом. Они не только предлагают свободный доступ к своему исходному коду, который можно просматривать и проводить с ним эксперименты, но представляют собой полноценные проекты с открытым кодом. Запросы на изменение принимались и фактически даже ожидались. Вклад со стороны сообщества разработчиков был весьма существенным — в первоначальном выпуске .NET Core участвовало более 10 000 разработчиков. То, что было относительно небольшой командой разработчиков в Microsoft, строящей .NET и связанные инфраструктуры, превратилось в крупное сообщество по разработке, которое теперь способно предоставлять дополнительную функциональность, улучшения производительности и исправления дефектов. Такая инициатива открытого кода охватывает не только программное обеспечение, но и документацию тоже. В действительности все ссылки на документацию, которые приводились в предшествующих главах, указывали на новую платформу документации с открытым кодом, находящуюся по адресу `http://docs.microsoft.com`.

В настоящей главе предлагается введение в философию .NET Core. В последующих трех главах раскрываются Entity Framework Core (глава 32) и ASP.NET Core (главы 33 и 34).

От проекта Project K до .NET Core

Как отметил Рич Лэндер в блоге .NET Blog (`https://blogs.msdn.microsoft.com/dotnet/2016/06/27/announcing-net-core-1-0/`), в Microsoft получили значительное количество запросов относительно функционирования ASP.NET на платформах, отличающихся от Windows. Исходя из этих запросов и параллельных обсуждений с командой разработчиков для Windows, касающихся Windows Nano, родился проект Project K, ориентированный на дополнительные платформы.

Начальные усилия были сосредоточены на ASP.NET. Для запуска под управлением других ОС требовалось устранить зависимость от `System.Web`, что было нетривиальным и по существу привело бы к переписыванию инфраструктуры. Нужно было принять решение о том, что будет включено. При исследовании разных платформ ASP.NET (Web Forms, ASP.NET MVC и ASP.NET Web API) выяснилось, что наилучшим подходом

было создание одной платформы, которая могла бы использоваться для веб-приложений и служб REST. Платформы MVC и Web API в конечном итоге были объединены в одну инфраструктуру (вместо того, чтобы быть тесно связанными родственниками как в полной платформе .NET Framework). Новая платформа для разработки веб-приложений сначала называлась ASP.NET 5.

Новой веб-платформе понадобился уровень доступа к данным, и очевидным выбором была инфраструктура Entity Framework. По существу все также сводилось к переписыванию версии EF 6, и было выбрано начальное название Entity Framework 7.

Откровенно говоря, по нашему мнению это наихудшие названия, какие только можно было выбрать. Трудно сказать, сколько времени было потрачено нами на объяснение заказчикам и участникам конференций, что ASP.NET 5 не является следующей версией полной платформы .NET Framework, даже если 5 больше 4.6 (в тот момент текущая версия полной платформы .NET Framework), а EF 7 — не следующая версия EF, даже если 7 больше 6.

Хорошая новость в том, что недоразумения были устранены, а названия межплатформенных версий .NET получили добавку Core, как в .NET Core, ASP.NET Core и Entity Framework Core. Такое разграничение привело в порядок большую часть неразберихи (хоть и не всю), связанной с различными инфраструктурами и их местом в экосистеме разработки.

Будущее полной платформы .NET Framework

Что можно сказать о полной платформе .NET Framework и связанных инфраструктурах? Поддерживаются ли они? Придется ли переписывать все ранее созданные приложения до того, как поддержка прекратится? По обоим пунктам — безусловно, нет! Данная книга должна развеять любые сомнения относительно продолжения разработки C# и .NET. Если бы платформа .NET не обновлялась, то книга не добралась бы до 8-го издания, раскрывающего C# 7.1 и .NET 4.7.

Продолжают поступать дополнительные вопросы о будущем разнообразных инфраструктур, таких как Windows Forms, Web Forms, WPF, WFC и т.д. Надо сказать, каких-то потрясающих новостей, связанных с многочисленными инфраструктурами .NET, было не сильно много, но это вовсе не означает, что работа над ними была прекращена. Можно утверждать, что такие технологии “закончены”. Инфраструктуры Windows Forms и Web Forms все еще широко применяются в наши дни, и отсутствие описания упомянутых инфраструктур в основных главах книги совершенно не означает, что мы не верим в их жизнеспособность; просто нам нечего сообщать сверх того, что приводилось в предшествующих изданиях.

На заметку! По-прежнему остается вопрос, какой будет следующая старшая версия полной платформы .NET. Логичным номером следующей версии является 5, но, к сожалению, данная карта сыграна. Мы уверены, что со временем все прояснится, а настоящая книга поможет понять текущее положение дел.

Цели .NET Core

Запуск под управлением ОС, отличающихся от Windows, был достаточно высокой целью для новой платформы, но далеко не единственной. Примерно с 2002 года .NET существовала как производственная инфраструктура, и с той поры в нее были внесены многочисленные изменения. Разработчики стали умнее, компьютеры — быстрее, требования пользователей возросли, а мобильность превратилась в доминирующую силу (и это лишь несколько аспектов).

Ниже перечислены некоторые цели .NET Core.

- *Межплатформенная поддержка*. Платформа .NET Core способна функционировать в средах Windows, Linux и macOS. Приложения .NET Core могут строиться на разных платформах с помощью Visual Studio Code или Visual Studio для Mac. Кроме того, включение Xamarin добавляет iOS и Android к числу платформ, поддерживаемых для развертывания.
- *Производительность*. Производительность .NET Core постоянно приближается к вершине всех важных графиков производительности, и в каждый выпуск вносятся дополнительные усовершенствования.
- *Переносимые библиотеки классов*, пригодные к потреблению всеми исполняющими средами .NET. В .NET Core введен стандарт .NET Standard — формальная спецификация, устанавливающая согласованное поведение исполняющей среды .NET.
- *Переносимое или автономное развертывание*. Приложения .NET Core могут развертываться параллельно с инфраструктурой либо использовать установленную копию .NET Core на уровне машины.
- *Полная поддержка командной строки*. Платформа .NET Core сосредоточена на полной поддержке командной строки в качестве основной цели.
- *Открытый код*. Как упоминалось, платформа .NET Core и ее документация являются системами с открытым кодом, укомплектованными запросами на изменение от мирового сообщества разработчиков.
- *Возможность взаимодействия с полной платформой .NET Framework*. Выпуск 2.0 платформы .NET Core разрешает ссылаться на библиотеки .NET Framework.

Мы уверены, что разработчики из Microsoft просмотрели существующую кодовую базу полной платформы .NET Framework и с учетом заявленных целей осознали невозможность изменения полной платформы .NET Framework при сохранении работоспособности всех производственных приложений. Явно требовалось целиком переписать код платформы .NET, а не только ASP.NET.

Давайте теперь рассмотрим каждую цель подробнее.

Межплатформенная поддержка

Ранее говорилось, что приложения .NET Core не ограничиваются выполнением в средах ОС, основанных на Windows. Это открывает широкий спектр вариантов для развертывания (и разработки) приложений с применением .NET и C#. Первоначальный выпуск .NET Core поддерживал ОС Windows (само собой разумеется), macOS и несколько дистрибутивов Linux, а также iOS и Android посредством Xamarin. В каждый последующий выпуск включались новые дистрибутивы и расширенная поддержка версий Linux. Список поддерживаемых ОС доступен по адресу `https://github.com/dotnet/core/blob/master/roadmap.md#technology-roadmaps`.

Вас может интересовать, каким образом межплатформенная версия .NET может помочь. В конце концов, возможно, вы уже посвятили себя ОС Windows и разработке на машине Windows (или на машине Mac с использованием технологии виртуализации).

Разработка приложений .NET Core где угодно

Имея дело с .NET Core, строить мобильные и веб-приложения можно где угодно. Среда Visual Studio по-прежнему выполняется на машине Windows (конечно же), но теперь доступна версия Visual Studio для Mac, так что разрабатывать приложения .NET Core можно прямо на Mac. А благодаря основанной на файлах системе проектов (которая вскоре будет раскрыта), появляется возможность применения при разработке приложений .NET Core межплатформенных инструментов вроде Visual Studio Code.

Дополнительные варианты развертывания

Наряду с тем, что опытные разработчики .NET уже могли запускать свои приложения под управлением Windows, платформа .NET Core привносит дополнительные варианты развертывания. Поддерживать серверы Linux обычно дешевле, чем серверы Windows, особенно в облаке, так что наличие возможности развертывания веб-приложений и служб в Linux может вылиться в существенную экономию финансовых средств. Главным образом это благоприятно для стартапов и компаний, желающих сократить количество серверов Windows.

Контейнеризация

В дополнение к поддержке дистрибутивов Linux платформа .NET Core поддерживает контейнеризацию приложений .NET Core. Популярные поставщики контейнеров наподобие Docker значительно уменьшили сложность выпуска приложений в разных средах (например, от разработки до тестирования). Приложение и необходимые файлы времени выполнения (включая ОС) упаковываются в один контейнер, который затем копируется из одной среды в другую, не беспокоясь об установке. Благодаря добавлению в Windows и Azure поддержки Docker положение дел стало еще лучше.

Теперь вы можете вести разработку в контейнере Docker и по готовности приложения к интеграционному тестированию просто перемещать контейнер из одной среды в другую. Никакого процесса установки или сложного развертывания! Когда приложение готово для эксплуатации в производственной среде, вы всего лишь переносите контейнер на производственный сервер. Проблемы с выпуском производственных версий, когда приложения отказывались функционировать в целевой среде, хотя разработчики гордо заявляли о том, что они успешно работали на их машинах, остались в прошлом.

Производительность

Приложения ASP.NET Core постоянно приближаются к вершине всех эталонных тестов. Причина в том, что в рамках проектных целей производительность считается “полноправным гражданином”. За прошедшие пятнадцать лет полная платформа .NET Framework крайне разрослась, поэтому выжать максимальную производительность из трудного в изменении кода в лучшем случае нелегко.

Поскольку .NET Core в значительной степени является переписанной версией платформы и инфраструктур, у ее создателей была возможность заблаговременно обдумать вопросы оптимизации и помнить о них во время разработки, а не после того, как платформа готова. Здесь самое важное кроется в слове “заблаговременно”. Производительность теперь — неотъемлемая часть архитектурных и проектных решений.

Переносимые библиотеки классов, соответствующие стандарту .NET Standard

Стандарт .NET Standard представляет собой формальную спецификацию для API-интерфейсов .NET, которые доступны всем исполняющим средам .NET. Он предназначен для обеспечения большей согласованности в экосистеме .NET. Благодаря стандарту .NET Standard становятся возможными следующие основные сценарии.

- Определение унифицированного набора API-интерфейсов в библиотеке базовых классов для всех реализаций .NET независимо от рабочей нагрузки.
- Появление у разработчиков возможности производить переносимые библиотеки, пригодные к потреблению различными реализациями .NET.
- Сокращение (или устранение) условной компиляции разделяемых ресурсов.

Любая сборка, нацеленная на специфическую версию .NET Standard, будет доступна любой другой сборке, нацеленной на ту же самую версию .NET Standard. Разные реализации .NET нацелены на специфические версии .NET Standard. С дополнительными сведениями о стандарте .NET Standard, а также о совместимости между платформами, инфраструктурами и версиями можно ознакомиться по адресу `https://docs.microsoft.com/ru-ru/dotnet/standard/net-standard`.

Переносимые или автономные модели развертывания

В отличие от полной версии .NET Framework платформа .NET Core поддерживает подлинную установку бок о бок. Установка добавочных версий .NET Core не затрагивает существующие установленные версии или приложения. В итоге спектр моделей развертывания для приложений .NET Core расширяется.

При переносимой модели развертывания приложение конфигурируется согласно версии .NET Core, установленной на целевой машине, а в данные развертывания включаются только пакеты, специфичные для приложения. Такой подход обеспечивает небольшой размер установочного пакета, но требует наличия на машине установленной копии целевой версии инфраструктуры.

При автономной модели развертывания пакет содержит все файлы приложения, а также обязательные файлы CoreFX и CoreCLR для целевой платформы. Такой подход очевидным образом увеличивает размер установочного пакета, но изолирует приложение от любых проблем, связанных с машиной (таких как отсутствие нужной версии .NET Core).

Полная поддержка командной строки

Понимая, что при разработке приложений .NET Core далеко не все будут использовать Visual Studio, в команде приняли решение сконцентрироваться на поддержке командной строки, прежде чем снабжать инструментами Visual Studio. В каждом выпуске до .NET Core 2.0 инструментарий отставал от того, что можно было делать с помощью текстового редактора и интерфейса командной строки .NET Core.

Инструментарий совершил большой шаг в своем развитии с момента выхода .NET Core 1.0 (когда он даже не был RTM), но все еще отстает от инструментальных средств Visual Studio, доступных для разработки приложений .NET Framework.

Открытый код

Как уже упоминалось, компания Microsoft выпустила платформу .NET Core и связанные инфраструктуры как подлинные системы с открытым кодом. Над первоначальным выпуском трудилось свыше 10 000 участников, и их число продолжает расти.

Возможность взаимодействия с .NET Framework

С выходом Visual Studio 15.3 и .NET Core 2.0 появилась возможность ссылаться из библиотек .NET Standard на библиотеки .NET Framework. Такое обновление улучшает механизм для переноса кода существующих приложений на платформу .NET Core.

Разумеется, есть ряд ограничений. В сборках, на которые можно ссылаться, должны применяться только типы, поддерживаемые стандартом .NET Standard. Хотя у вас может получиться работать со сборкой, использующей API-интерфейсы, которые отсутствуют в .NET Standard, данный сценарий не является поддерживаемым и должен сопровождаться всесторонним тестированием.

Состав .NET Core

В общих терминах платформа образована из четырех основных частей:

- исполняющая среда .NET Core;
- набор библиотек инфраструктуры;
- инструменты SDK и хост приложений `dotnet`;
- компиляторы языков.

Исполняющая среда .NET Core (CoreCLR)

Это базовая библиотека для .NET Core. Она включает сборщик мусора, компилятор JIT, базовые типы .NET и множество низкоуровневых классов. Исполняющая среда предоставляет мост между библиотеками инфраструктуры .NET Core (CoreFX) и лежащими в основе ОС. В ее состав включены только типы, которые имеют строгую зависимость от внутренней работы исполняющей среды. Большая часть библиотеки классов реализована в виде независимых пакетов NuGet.

При проектировании CoreCLR разработчики пытались свести к минимуму объем реализованного кода, оставляя специфические реализации многих классов инфраструктуры на CoreFX. В результате получилась небольшая гибкая кодовая база, которую можно модифицировать и быстро развертывать для исправления дефектов или добавления функциональных средств. Сама по себе среда CoreCLR делает не особо много работы. Любой определенный в ней библиотечный код скомпилирован в сборку `System.Private.CoreLib.dll`, которая не предназначена для потребления за пределами CoreCLR или CoreFX.

В состав дополнительного инструментария, предлагаемого CoreCLR, входит ILDASM и ILASM (версии .NET Core программ, которые вы применяли ранее в книге), а также хост тестирования — небольшая оболочка для запуска DLL-библиотек IL из командной строки.

Библиотеки инфраструктуры (CoreFX)

Это набор фундаментальных библиотек, включающий классы для коллекций, файловых систем, консоли, разметки XML, асинхронной работы и многих других элементов. Библиотеки инфраструктуры построены поверх CoreCLR и предоставляют для других инфраструктур интерфейсные точки в исполняющую среду. Помимо специфических реализаций CoreCLR, содержащихся в CoreFX, остальные библиотеки имеют независимую от исполняющей среды и платформы природу. В CoreCLR находится `mscorlib.dll` — открытый фасад CoreCLR. Вместе CoreCLR и CoreFX образуют .NET Core.

Инструменты SDK и хост приложений `dotnet`

Включенные в SDK инструменты представляют собой интерфейс командной строки (command-line interface — CLI) платформы .NET, используемый для построения приложений и библиотек .NET Core. Хост приложений `dotnet` является универсальным драйвером для выполнения команд CLI, включая приложения .NET Core.

Команды CLI и приложения .NET Core запускаются с применением хоста приложений `dotnet`. Чтобы увидеть его в действии, необходимо открыть окно командной строки (оно не обязательно должно быть командной подсказкой для разработчиков из Visual Studio) и ввести следующую команду:

```
dotnet --version
```

Команда запускает хост приложений `dotnet` и отображает установленную в текущий момент и доступную через переменную среды `PATH` версию .NET Core SDK. С помощью CLI можно вводить много готовых команд, самые распространенные из которых перечислены в табл. 31.1.

Таблица 31.1. Распространенные команды CLI в .NET Core

Команда	Описание
`--help`	Выводит справочную информацию
`--version`	Выводит версию .NET Core SDK
`--info`	Выводит версию инструментов командной строки .NET Core и разделяемого хоста инфраструктуры
`new`	Инициализирует пример консольного приложения .NET Core
`restore`	Восстанавливает зависимости для заданного приложения. В версии 2.0 неявно выполняется с командами `build` и `run`
`build`	Строит приложение .NET Core
`clean`	Очищает вывод проекта
`publish`	Публикует переносимое или независимое приложение .NET
`run`	Запускает приложение из исходного кода
`test`	Прогоняет тесты, используя указанное средство выполнения тестов
`pack`	Создает пакет NuGet из исходного кода

В дополнение к встроенным командам CLI инфраструктуры могут добавлять собственные команды. Например, инфраструктура Entity Framework Core вводит серию команд для миграций и обновлений базы данных, которые будут применяться в следующей главе.

Компиляторы языков

Платформа .NET Compiler ("Roslyn") предоставляет компиляторы с открытым кодом языков C# и Visual Basic с развитыми API-интерфейсами для анализа кода. Несмотря на то что в выпуск .NET Core 2.0 была добавлена *некоторая* поддержка Visual Basic, язык C# по-прежнему считается основным и поддерживается во всех инфраструктурах .NET Core.

Жизненный цикл поддержки .NET Core

Платформа .NET Core поддерживается компанией Microsoft, хотя она и с открытым кодом. Каждый выпуск имеет определенный жизненный цикл и разделяется на две категории: долгосрочная поддержка (Long Term Support — LTS) и текущая (Current), ранее называемая ускоренной поддержкой (Fast Track Support). Каждый старший или младший выпуск, как правило, будет поддерживаться на протяжении трех лет, пока исправления сохраняются текущими (скажем, 1.0.1), с несколькими исключениями, которые обсуждаются ниже.

Выпуски LTS:

- представляются старшими номерами версий (например, 1.0, 2.0);
- поддерживаются в течение трех лет после общей доступности (general availability — GA) выпуска LTS;
- или поддерживаются в течение одного года после следующего выпуска LTS (какой бы он ни был коротким).

Выпуски Current:

- представляются младшими номерами версий (например, 1.1, 1.2);
- поддерживаются в рамках того же самого трехлетнего окна, как и родительский выпуск LTS;
- поддерживаются в течение трех месяцев после общей доступности последующего выпуска Current;
- или поддерживаются в течение одного года после общей доступности последующего выпуска LTS.

Если все выглядит сложным и запутанным, то не рассчитывайте найти у нас какие-то аргументы против этого. Лицензирование продуктов Microsoft никогда не было простым, и .NET Core ничем в данном плане не отличается. Отправьте свою команду юристов по адресу `https://www.microsoft.com/net/core/support` за дополнительной информацией и обеспечьте соблюдение всех условий.

На заметку! В пояснительной записке к выпуску .Net Core 2.0 упоминалось, что версия .NET Core 1.1 присоединилась к .NET Core 1.0 как продукт с поддержкой LTS, а не Current.

Установка .NET Core 2.0

Если вы еще не установили продукт Visual Studio 2017, тогда установите в соответствие с инструкциями из главы 2 (по крайней мере) такие рабочие нагрузки:

- .NET desktop development (Разработка настольных приложений .NET);
- ASP.NET and web development (Разработка приложений ASP.NET и веб-приложений);
- .NET Core cross-platform development (Разработка межплатформенных приложений .NET Core).

Для использования .NET Core 2.0 требуется располагать, по меньшей мере, версией 15.3 продукта Visual Studio 2017. Вдобавок на время написания настоящей главы комплект .NET Core 2.0 SDK должен был устанавливаться отдельно и загружаться по адресу `https://dot.net/core`.

После установки Visual Studio 15.3 и .NET Core 2.0 SDK нужно открыть окно командной строки и ввести следующую команду:

```
dotnet --version
```

В окне консоли должна отобразиться версия 2.0.0 (или выше, что зависит от того, когда вы читаете эту книгу). Затем понадобится ввести команду:

```
dotnet --info
```

В окне консоли должны быть выведены такие данные:

```
.NET Command Line Tools (2.0.0)

Product Information:
 Version:            2.0.0
 Commit SHA-1 hash:  cdcd1928c9

Runtime Environment:
 OS Name:       Windows
 OS Version:    10.0.16257
 OS Platform:   Windows
 RID:           win10-x64
 Base Path:     C:\Program Files\dotnet\sdk\2.0.0\

Microsoft .NET Core Shared Framework Host

 Version   :    2.0.0
 Build     :    e8b8861ac7faf042c87a5c2f9f2d04c98b69f28d
```

Сведения подтвердят, что на машине установлены подходящие версии для финальных глав книги.

Сравнение с полной платформой .NET Framework

Наряду с тем, что в .NET Core применяются язык C# и многие из тех же API-интерфейсов, которые присутствуют в полной платформе .NET Framework, существует несколько важных отличий:

- расширенная поддержка платформы;
- открытый код;
- сниженное количество поддерживаемых моделей приложений;
- меньшее число реализованных API-интерфейсов и подсистем.

Первые два отличия уже были раскрыты ранее в главе, а оставшиеся два рассматриваются в последующих разделах.

Сниженное количество поддерживаемых моделей приложений

Ранее упоминалось, что .NET Core не поддерживает все модели приложений .NET Framework, особенно те, которые построены прямо на основе технологий Windows. Кроме того, любое приложение .NET Core будет либо консольным приложением, либо библиотекой классов. Даже приложения ASP.NET Core являются просто консольными приложениями, которые создают веб-хост, как будет показано в главах 33 и 34.

Меньшее число реализованных API-интерфейсов и подсистем

Полная платформа .NET Framework совершенно статична и попытка изменить все API-интерфейсы одновременно была бы почти неразрешимой задачей. Следуя принципам гибкой разработки, создатели .NET Core определили, какой минимальный жизнеспособный продукт обеспечивал поддержку .NET Core 1.0, и выпустили именно то, чего достаточно для удовлетворения таких требований. Каждый последующий выпуск .NET Core включал многие дополнительные API-интерфейсы из полной платформы .NET Framework, причем выпуск 2.0 реализует свыше 32 000 API-интерфейсов.

Платформа .NET Core реализует только подмножество подсистем из полной платформы .NET Framework, преследуя цель предложить более простую реализацию и модель программирования. Например, наряду с тем, что рефлексия поддерживается, безопасность доступа кода (Code Access Security — CAS) — нет.

Резюме

В настоящей главе была заложена основа для последующих трех глав путем демонстрации сходных черт и отличий между полной платформой .NET Framework и .NET Core. Как будет показано в дальнейших главах, разработка приложений .NET Core похожа на разработку приложений .NET Framework. Приходится иметь дело с меньшим числом инфраструктур и API-интерфейсов, но в остальном это всего лишь код C#.

При проектировании .NET Core преследовались многие цели, и команда создателей (включая мировое сообщество с открытым кодом) за короткое время выполнила потрясающую работу, снабдив разработчиков приложений .NET замечательным набором инструментов, который можно использовать для межплатформенной разработки и развертывания.

На заметку! Последующие три главы, посвященные EF Core 2.0 (глава 32) и ASP.NET Core 2.0 (главы 33 и 34), ограничены по объему того, что можно было бы раскрыть. Более подробное изложение данных тем, а также особенностей применения инфраструктур JavaScript для построения пользовательских интерфейсов с помощью .NET Core можно найти в книге *Building Web Applications with Visual Studio 2017* (`https://www.amazon.com/Building-Applications-Visual-Studio-2017/dp/1484224779/`).
