5

Обзор .NET Framework

Почти все возможности .NET Framework доступны через обширное множество управляемых типов. Эти типы организованы в иерархические пространства имен и упакованы в набор сборок, которые вместе со средой CLR образуют платформу .NET.

Некоторые типы .NET используются напрямую CLR и являются критически важными для среды управляемого размещения. Такие типы находятся в сборке по имени *mscorlib.dll* и включают встроенные типы C#, а также базовые классы коллекций, типы для обработки потоков данных, сериализации, рефлексии, многопоточности и собственной возможности взаимодействия ("mscorlib" представляет собой аббревиатуру от Multi-language Standard Common Object Runtime Library — стандартная многоязыковая общая объектная библиотека времени выполнения).

Уровнем выше находятся дополнительные типы, которые расширяют функциональность уровня CLR, предоставляя такие средства, как XML, работа в сети и LINQ. Они находятся в сборках *System.dll*, *System.Xml.dll* и *System.Core.dll*, и совместно с mscorlib.dll формируют развитую среду для программирования, на основе которой построены остальные части .NET Framework. Такая "центральная инфраструктура" в значительной степени определяет контекст оставшихся глав настоящей книги.

Остаток инфраструктуры .NET Framework состоит из прикладных API-интерфейсов, большинство из которых охватывают три области функциональности:

- технологии пользовательских интерфейсов;
- технологии серверной части;
- технологии распределенных систем.

В табл. 5.1 показана хронология совместимости между версиями C#, CLR и .NET Framework.

В главе приведен обзор всех ключевых областей .NET Framework, начиная с раскрытых в настоящей книге основных типов и заканчивая краткими сведениями о прикладных технологиях.

Таблица 5.1. Версии C#, CLR и .NET Framework

Версия C#	Версия CLR	Версии .NET Framework
1.0	1.0	1.0
1.2	1.1	1.1
2.0	2.0	2.0, 3.0
3.0	2.0 (SP2)	3.5
4.0	4.0	4.0
5.0	4.5 (исправленная версия CLR 4.0)	4.5
6.0	4.6 (исправленная версия CLR 4.0)	4.6
7.0	4.6/4.7 (исправленная версия CLR 4.0)	4.6/4.7

Нововведения версии .NET Framework 4.6

- Сборщик мусора (Garbage Collector — GC) предлагает больший контроль над тем, когда (не) производить сборку мусора, через новые методы класса `GC`. При вызове метода `GC.Collect` также доступны дополнительные параметры для более точной настройки.
- Появился совершенно новый более быстрый 64-разрядный JIT-компилятор.
- Пространство имен `System.Numerics` теперь включает аппаратно-ускоренные типы матриц, векторов, `BigInteger` и `Complex`.
- Появился новый класс `System.AppContext`, который снабжает авторов библиотек согласованным механизмом, позволяющим потребителям библиотек включать или отключать средства новых API-интерфейсов.
- Объекты `Task` при создании выбирают культуру текущего потока и культуру пользовательского интерфейса.
- Интерфейс `IReadOnlyCollection<T>` реализует большее количество типов коллекций.
- В инфраструктуру WPF внесены дополнительные усовершенствования, включая улучшенную обработку касаний и более высоких DPI.
- Инфраструктура ASP.NET теперь поддерживает HTTP/2 и протокол привязки по маркерам (Token Binding Protocol) в Windows 10.

Сборки и пространства имен в .NET Framework *пересекаются*. Наиболее экстремальными примерами считаются `mscorlib.dll` и `System.Core.dll`, где определены типы в десятках пространств имен, причем ни одно из них не начинается с `mscorlib` или `System.Core`. Однако менее очевидные случаи являются более запутанными, такие как типы в `System.Security.Cryptography`. Большинство типов в этом пространстве имен находится в сборке `System.dll` за исключением нескольких типов, которые расположены в сборке `System.Security.dll`.

На веб-сайте, посвященном книге, содержится полное отображение пространств имен .NET Framework на сборки (`http://www.albahari.com/nutshell/NamespaceReference.aspx`).

Многие ключевые типы определены в сборках *mscorlib.dll*, *System.dll* и *System.Core.dll*. Первая сборка, *mscorlib.dll*, содержит типы, которые требуются для самой исполняющей среды; в сборках *System.dll* и *System.Core.dll* находятся дополнительные типы, необходимые для программистов. Причина того, что последние две сборки являются отдельными, чисто историческая: версию .NET Framework 3.5 в Microsoft старались сделать насколько возможно *добавочной*, т.к. она запускалась в виде уровня поверх существующей среды CLR 2.0. Вследствие этого почти все новые основные типы (такие как классы, поддерживающие LINQ) вошли в новую сборку, которую в Microsoft назвали *System.Core.dll*.

Нововведения версии .NET Framework 4.7

Версия .NET Framework 4.7 — в большей степени корректировочный, нежели вводящий новые средства выпуск, с многочисленными исправлениями дефектов и незначительными улучшениями. Кроме того, данная версия обладает следующими особенностями.

- Структура `System.ValueTuple` входит в состав .NET Framework 4.7, так что можно использовать кортежи C# 7, не ссылаясь на сборку *System.ValueTuple.dll*.
- В WPF улучшилась поддержка касаний.
- В Windows Forms улучшилась поддержка мониторов с высокими параметрами DPI.

.NET Standard 2.0

В главе 1 мы описали три главные альтернативы .NET Framework для межплатформенной разработки:

- UWP для устройств и настольных компьютеров Windows 10;
- .NET Core/ASP.NET Core для Windows, Linux и MacOS;
- Xamarin для мобильных устройств (с iOS, Android и Windows 10).

Хорошая новость в том, что с выходом .NET Core 2.0 указанные инфраструктуры — наряду с .NET Framework 4.6.1 и последующими версиями — сблизились по своей основной функциональности и теперь все они предлагают библиотеку базовых классов (BCL) с похожими типами и членами. Такая общность была формализована как стандарт под названием *.NET Standard 2.0*.

При написании библиотеки в Visual Studio 2017 вместо специфической инфраструктуры вы можете выбрать в качестве целевой платформы *.NET Standard 2.0*. Тогда ваша библиотека станет *переносимой*, и та же самая сборка будет запускаться без модификаций под управлением (современных версий) всех четырех инфраструктур.

.NET Standard — не .NET Framework; это просто спецификация, описывающая минимальный уровень функциональности (типы и члены), который гарантирует совместимость с определенным набором инфраструктур. Концепция похожа на интерфейсы C#: стандарт .NET Standard подобен интерфейсу, который конкретные типы (инфраструктуры) могут реализовать.

В настоящей книге раскрывается большинство из того, что находится в .NET Standard 2.0.

Старые стандарты .NET Standard

Существуют и применяются также старые стандарты .NET Standard, наиболее примечательны из которых 1.1, 1.2, 1.3 и 1.6. Стандарт с более высоким номером всегда представляет собой строгое надмножество стандарта с более низким номером. Например, при написании библиотеки, которая нацелена на .NET Standard 1.6, вы будете поддерживать не только последние версии четырех крупных инфраструктур, но также .NET Core 1.0. А если вы нацеливаете библиотеку на .NET Standard 1.3, то будет поддерживаться все, что мы уже упомянули, плюс .NET Framework 4.6.0 (табл. 5.2).

Таблица 5.2. Старые стандарты .NET Standard

Если вы нацеливаете библиотеку на…	То будут также поддерживаться…
.NET Standard 1.6	.NET Core 1.0
.NET Standard 1.3	Указанное выше плюс .NET Framework 4.6.0
.NET Standard 1.2	Указанное выше плюс .NET Framework 4.5.1, Windows Phone 8.1, WinRT для Windows 8.1
.NET Standard 1.1	Указанное выше плюс .NET Framework 4.5.0, Windows Phone 8.0, WinRT для Windows 8.0

В стандартах 1.x отсутствуют тысячи API-интерфейсов, которые находятся в стандарте 2.0, в том числе большинство того, что мы описываем в этой книге. В результате нацеливание на какой-то стандарт 1.x становится гораздо более затруднительным, особенно в случае необходимости интеграции с существующим кодом или библиотеками.

Если вас интересует поддержка более старых инфраструктур, но не межплатформенная совместимость, тогда лучше нацеливаться на старую версию специфической инфраструктуры. В случае Windows удачным выбором будет версия .NET Framework 4.5, т.к. она широко развернута (заранее установлена на всех машинах с Windows 8 и выше) и содержит бо́льшую часть того, что есть в .NET Framework 4.7.

Вы можете думать о стандарте .NET Standard как о наименьшем общем знаменателе. В случае стандарта .NET Standard 2.0 четыре инфраструктуры, которые ему следуют, имеют похожую библиотеку базовых классов, поэтому наименьший общий знаменатель является крупным и полезным. Тем не менее, если вам также нужна совместимость с инфраструктурой .NET Core 1.0 (с ее значительно усеченной библиотекой BCL), тогда наименьший общий знаменатель — .NET Standard 1.x — становится гораздо *уже* и менее полезным.

Ссылочные сборки

При компиляции программы вы обязаны ссылаться на сборки, которые содержат части инфраструктуры, потребляемые вашей программой. Например, простая консольная программа для .NET Framework, которая включает запрос LINQ to XML, потребовала бы сборки `mscorlib.dll`, `System.dll`, `System.Xml.dll`, `System.Xml.Linq.dll` и `System.Core.dll`.

В среде Visual Studio это делается путем добавления к проекту ссылок (только что перечисленные ссылки добавляются автоматически, когда создается проект, нацеленный на .NET Framework 4.x). Однако сборки, на которые производится ссылка, должны существовать только в интересах компилятора и не обязательно будут теми же, что используются во время выполнения. Следовательно, допустимо применять специальные ссылочные сборки, которые существуют как пустые оболочки без какого-либо скомпилированного кода. Именно так работает стандарт .NET Standard: вы добавляете *ссылочную сборку* по имени `netstandard.dll`, которая содержит все допустимые типы и члены в .NET Standard 2.0 (но не фактический скомпилированный код). Затем посредством атрибутов переадресации сборок во время выполнения загружаются "реальные" сборки. (Выбор "реальных" сборок будет зависеть от того, под управлением какой инфраструктуры происходит запуск.)

Ссылочные сборки также позволяют нацеливаться на более низкую версию .NET Framework, которая установлена на вашей машине. Скажем, если вы установили .NET Framework 4.7 вместе с Visual Studio 2017, то по-прежнему можете нацеливать свой проект на .NET Framework 4.0. Благодаря набору ссылочных сборок .NET Framework 4.0 ваш проект будет способен видеть только типы/члены .NET Framework 4.0.

Среда CLR и ядро платформы

Системные типы

Наиболее фундаментальные типы находятся прямо в пространстве имен `System`. В их число входят встроенные типы C#, базовый класс `Exception`, базовые классы `Enum`, `Array` и `Delegate`, а также типы `Nullable`, `Type`, `DateTime`, `TimeSpan` и `Guid`. Кроме того, пространство имен `System` включает типы для выполнения математических функций (`Math`), генерации случайных чисел (`Random`) и преобразования между различными типами (`Convert` и `BitConverter`).

Фундаментальные типы описаны в главе 6 вместе с интерфейсами, которые определяют стандартные протоколы, используемые повсеместно в .NET Framework для решения таких задач, как форматирование (`IFormattable`) и сравнение порядка (`IComparable`).

В пространстве имен `System` также определен интерфейс `IDisposable` и класс `GC` для взаимодействия со сборщиком мусора. Эти темы будут раскрыты в главе 12.

Обработка текста

Пространство имен `System.Text` содержит класс `StringBuilder` (редактируемый или *изменяемый* родственник `string`) и типы для работы с кодировками текста, такими как UTF-8 (`Encoding` и его подтипы). Мы рассмотрим их в главе 6.

Пространство имен `System.Text.RegularExpressions` содержит типы, которые выполняют расширенные операции поиска и замены на основе шаблона; такие типы описаны в главе 26.

Коллекции

Платформа .NET Framework предлагает разнообразные классы для управления коллекциями элементов. Они включают структуры, основанные на списках и словарях, и работают в сочетании с набором стандартных интерфейсов, которые унифицируют их общие характеристики. Все типы коллекций определены в следующих пространствах имен, описанных в главе 7:

```
System.Collections                // Необобщенные коллекции
System.Collections.Generic        // Обобщенные коллекции
System.Collections.Specialized    // Строго типизированные коллекции
System.Collections.ObjectModel    // Базовые типы для создания собственных
                                  // коллекций
System.Collections.Concurrent     // Коллекции, безопасные в отношении
                                  // потоков (глава 23)
```

Запросы

Язык интегрированных запросов (Language Integrated Query – LINQ) появился в версии .NET Framework 3.5. Язык LINQ позволяет выполнять безопасные в отношении типов запросы к локальным и удаленным коллекциям (например, таблицам SQL Server); он описан в главах 8–10. Крупное преимущество языка LINQ в том, что он предоставляет согласованный API-интерфейс запросов для разнообразных предметных областей. Основные типы находятся в перечисленных ниже пространствах имен и являются частью стандарта .NET Standard 2.0:

```
System.Linq              // LINQ to Objects и PLINQ
System.Linq.Expressions  // Для ручного построения выражений
System.Xml.Linq          // LINQ to XML
```

Полная инфраструктура .NET Framework также включает следующие пространства имен, которые будут описаны в разделе "Технологии серверной части" далее в главе:

```
System.Data.Linq         // LINQ to SQL
System.Data.Entity       // LINQ to Entities (Entity Framework)
```

XML

Язык XML широко применяется внутри .NET Framework, поэтому поддерживается повсеместно. В главе 10 внимание сосредоточено целиком на LINQ to XML – легковесной объектной модели документа XML, которую можно конструировать и опрашивать с помощью LINQ. В главе 11 описана более старая модель W3C DOM, а также высокопроизводительные низкоуровневые классы для чтения/записи плюс поддержка .NET Framework для схем XML, стилевых таблиц и XPath. Ниже перечислены пространства имен, связанные с XML:

```
System.Xml               // XmlReader, XmlWriter и старая модель W3C DOM
System.Xml.Linq          // Объектная модель документа LINQ to XML
System.Xml.Schema        // Поддержка для XSD
System.Xml.Serialization // Декларативная сериализация XML для типов .NET
System.Xml.XPath         // Язык запросов XPath
System.Xml.Xsl           // Поддержка стилевых таблиц
```

Диагностика

В главе 13 мы раскроем возможности .NET по регистрации в журнале и утверждениям, а также покажем, как взаимодействовать с другими процессами, выполнять запись в журнал событий Windows и использовать счетчики производительности для проведения мониторинга. Соответствующие типы определены в пространстве имен `System.Diagnostics` и его подпространствах. Средства, специфичные для Windows, не входят в стандарт .NET Standard и доступны только в .NET Framework.

Параллелизм и асинхронность

Многим современным приложениям в каждый момент времени приходится иметь дело с несколькими действиями. С выходом версии C# 5.0 решение стало проще за счет асинхронных функций и таких высокоуровневых конструкций, как задачи и комбинаторы задач. Все это подробно объясняется в главе 14, которая начинается с рассмотрения основ многопоточности. Типы для работы с потоками и асинхронными операциями находятся в пространствах имен `System.Threading` и `System.Threading.Tasks`.

Потоки данных и ввод-вывод

Инфраструктура .NET Framework предоставляет потоковую модель для низкоуровневого ввода-вывода. Потоки данных обычно применяются для чтения и записи напрямую в файлы и сетевые подключения и могут соединяться в цепочки либо помещаться внутрь декорированных потоков с целью добавления функциональности сжатия или шифрования. В главе 15 описана потоковая архитектура .NET, а также специфическая поддержка для работы с файлами и каталогами, сжатием, изолированным хранилищем, каналами и файлами, отображенными в память. Тип `Stream` и типы ввода-вывода .NET определены в пространстве имен `System.IO` и его подпространствах, а типы WinRT для файлового ввода-вывода — в пространстве имен `Windows.Storage` и его подпространствах.

Работа с сетями

С помощью типов из пространства имен `System.Net` можно напрямую работать со стандартными сетевыми протоколами, такими как HTTP, FTP, TCP/IP и SMTP. В главе 16 мы покажем, как взаимодействовать с использованием каждого из упомянутых протоколов, начиная с простых задач вроде загрузки веб-страницы и заканчивая применением TCP/IP для извлечения электронной почты POP3. Ниже перечислены пространства имен, которые будут рассмотрены:

```
System.Net
System.Net.Http      // HttpClient
System.Net.Mail      // Для отправки электронной почты через SMTP
System.Net.Sockets   // TCP, UDP и IP
```

Последние два пространства имен не будут доступны для приложений Windows Store в случае нацеливания на Windows 8/8.1 (WinRT), но доступны для приложений Windows 10 Store (UWP) как часть контракта .NET Standard 2.0. В случае приложений WinRT для отправки электронной почты следует использовать библиотеки от независимых разработчиков, а для работы с сокетами — типы WinRT из пространства имен `Windows.Networking.Sockets`.

Сериализация

Инфраструктура .NET Framework предлагает несколько систем для сохранения и восстановления объектов в двоичном или текстовом представлении. Такие системы требуются в технологиях распределенных приложений, подобных WCF, Web Services и Remoting, а также применяются для сохранения и восстановления объектов из файлов. В главе 17 мы раскроем три важных механизма сериализации: сериализатор контрактов данных, двоичный сериализатор и сериализатор XML. (В .NET Framework теперь также доступен сериализатор JSON.)

Типы для сериализации находятся в следующих пространствах имен:

```
System.Runtime.Serialization
System.Xml.Serialization
```

Сборки, рефлексия и атрибуты

Сборки, в которые компилируются программы на C#, состоят из исполняемых инструкций (представленных на промежуточном языке (intermediate language — IL)) и метаданных, описывающих типы, члены и атрибуты программы. С помощью рефлексии можно просматривать метаданные во время выполнения и предпринимать действия вроде динамического вызова методов. Посредством пространства имен `Reflection.Emit` можно конструировать новый код на лету.

В главе 18 мы опишем строение сборок и их подписание, использование глобального кеша сборок (global assembly cache — GAC) и ресурсов, а также распознавание ссылок на файлы. В главе 19 мы раскроем рефлексию и атрибуты — покажем, как инспектировать метаданные, динамически вызывать функции, записывать специальные атрибуты, выпускать новые типы и производить разбор низкоуровневого кода IL. Типы для применения рефлексии и работы со сборками находятся в следующих пространствах имен:

```
System
System.Reflection
System.Reflection.Emit  // Только для .NET Framework
```

Динамическое программирование

В главе 20 мы рассмотрим несколько паттернов для динамического программирования и работы со средой DLR, которая стала частью CLR, начиная с версии .NET Framework 4.0. Мы покажем, как реализовать паттерн “Посетитель” (Visitor), создавать специальные динамические объекты и взаимодействовать с IronPython. Типы, предназначенные для динамического программирования, находятся в пространстве имен `System.Dynamic`.

Безопасность

Инфраструктура .NET Framework предоставляет собственный уровень безопасности, позволяя организовать работу в песочнице другим сборкам и себе самой. В главе 21 мы обсудим безопасность доступа кода, безопасность на основе удостоверений и ролей, а также модель прозрачности, введенную в CLR 4.0. Затем мы раскроем криптографические возможности .NET Framework, рассмотрев шифрование, хеширование и защиту данных. Типы для таких целей определены в следующих пространствах имен:

```
System.Security
System.Security.Permissions
System.Security.Policy
System.Security.Cryptography
```

Расширенная многопоточность

Асинхронные функции в C# значительно облегчают параллельное программирование, поскольку снижают потребность во взаимодействии с низкоуровневыми технологиями. Тем не менее, все еще возникают ситуации, когда нужны сигнальные конструкции, локальное хранилище потока, блокировки чтения/записи и т.д. Данные вопросы

подробно объясняются в главе 22. Типы, связанные с многопоточностью, находятся в пространстве имен `System.Threading`.

Параллельное программирование

В главе 23 мы рассмотрим библиотеки и типы для работы с многоядерными процессорами, включая API-интерфейсы для реализации параллелизма задач, императивного параллелизма данных и функционального параллелизма (PLINQ).

Домены приложений

Среда CLR предоставляет дополнительный уровень изоляции внутри процесса, который называется *доменом приложения*. В главе 24 мы исследуем свойства домена приложения, с которыми можно взаимодействовать, и покажем, как создавать и использовать дополнительные домены приложений в рамках одного и того же процесса для таких целей, как модульное тестирование. Мы также объясним, каким образом применять технологию Remoting для взаимодействия с доменами приложений.

Создание отдельных доменов приложений не является частью стандарта .NET Standard 2.0, хотя можно взаимодействовать с текущим доменом через класс `AppDomain` из пространства имен `System`.

Собственная возможность взаимодействия и возможность взаимодействия с COM

Существует возможность взаимодействия с собственным кодом, а также с кодом COM. Собственная возможность взаимодействия позволяет вызывать функции из неуправляемых DLL-библиотек, регистрировать обратные вызовы, отображать структуры данных и работать с собственными типами данных. Возможность взаимодействия с COM позволяет обращаться к типам COM и открывать для COM доступ к типам .NET. Типы, поддерживающие такую функциональность, определены в пространстве имен `System.Runtime.InteropServices` и рассматриваются в главе 25.

Прикладные технологии

Технологии пользовательских интерфейсов

Приложения, основанные на пользовательском интерфейсе, можно разделить на две категории: *тонкий клиент*, равнозначный веб-сайту, и *обогащенный клиент*, представляющий собой программу, которую конечный пользователь должен загрузить и установить на компьютере или на мобильном устройстве.

Для разработки приложений тонких клиентов платформа .NET предлагает инфраструктуры ASP.NET и ASP.NET Core.

Для построения приложений обогащенных клиентов, ориентированных на рабочий стол Windows 7/8/10, платформа .NET предоставляет API-интерфейсы WPF и Windows Forms. Для создания приложений обогащенных клиентов, нацеленных на iOS, Android и Windows Phone, имеется инфраструктура Xamarin, а для написания приложений обогащенных клиентов Windows Store, функционирующих на настольных компьютерах и устройствах Windows 10, доступна платформа UWP (см. табл. 1.1 в главе 1).

Наконец, существует гибридная технология под названием Silverlight, с которой практически перестали работать после выхода HTML5.

ASP.NET

Приложения, написанные с использованием ASP.NET, размещаются на сервере Windows IIS и могут быть доступны из любого веб-браузера. Ниже перечислены преимущества ASP.NET по сравнению с технологиями обогащенных клиентов.

- Отсутствует потребность в развертывании на клиентской стороне.
- Клиенты могут функционировать на платформах, отличных от Windows.
- Легко развертывать обновления.

Кроме того, поскольку бо́льшая часть кода, написанного в приложении ASP.NET, выполняется на сервере, уровень доступа к данным проектируется для функционирования в том же самом домене приложения — без ограничения безопасности или масштабируемости. Напротив, обогащенный клиент, который делает то же самое, обычно не настолько безопасный или масштабируемый. (В случае обогащенного клиента решение предусматривает создание *среднего уровня* между клиентом и базой данных. Средний уровень выполняется на удаленном сервере приложений (часто вместе с сервером базы данных) и взаимодействует с обогащенными клиентами через WCF, Web Services или Remoting.)

При написании своих веб-страниц вы можете выбирать между традиционным API-интерфейсом Web Forms и более новым API-интерфейсом MVC (Model-View-Controller — модель-представление-контроллер). Оба они построены на основе инфраструктуры ASP.NET. Технология Web Forms была частью .NET Framework с самого начала, а MVC реализована намного позже как реакция на успех Ruby on Rails и MonoRail. В целом инфраструктура MVC обеспечивает лучшую программную абстракцию по сравнению с Web Forms; она также позволяет иметь больший контроль над генерируемой HTML-разметкой. Единственное, в чем MVC проигрывает Web Forms — визуальный конструктор, что сохраняет Web Forms хорошим средством для построения веб-страниц с преимущественно статическим содержимым.

Ограничения ASP.NET в значительной степени отражают общие ограничения систем тонких клиентов.

- Несмотря на то что веб-браузер способен предложить насыщенный мощный интерфейс с помощью HTML5 и AJAX, в плане возможностей и производительности он по-прежнему уступает собственному API-интерфейсу обогащенного клиента.
- Поддержка состояния на стороне клиента — или от имени клиента — может быть громоздкой.

Типы, предназначенные для написания приложений ASP.NET, находятся в пространстве имен `System.Web.UI` и его подпространствах; они упакованы в сборку *`System.Web.dll`*. Инфраструктура ASP.NET доступна через NuGet.

ASP.NET Core

Относительно недавно добавленная инфраструктура ASP.NET Core похожа на ASP.NET, но функционирует в средах .NET Framework и .NET Core (делая возможным межплатформенное развертывание). Инфраструктура ASP.NET Core отличается легковесной модульной архитектурой, обладает способностью саморазмещения в специальном процессе и регламентируется лицензией для программного обеспечения с открытым кодом. В отличие от своих предшественников инфраструктура ASP.NET Core не зависит от `System.Web` и лишена исторического багажа Web Forms. Она особенно хорошо подходит для микросервисов и развертывания внутри контейнеров.

Windows Presentation Foundation (WPF)

Инфраструктура WPF была введена в .NET Framework 3.0 для написания приложений обогащенных клиентов. Ниже перечислены преимущества инфраструктуры WPF по сравнению с Windows Forms.

- Она поддерживает развитую графику, включая произвольные трансформации, трехмерную визуализацию, мультимедиа-возможности и подлинную прозрачность. Оформление поддерживается через стили и шаблоны.
- Основная единица измерения не базируется на пикселях, поэтому приложения корректно отображаются при любой настройке DPI (dots per inch — точек на дюйм).
- Она располагает обширной и гибкой поддержкой динамической компоновки, означающей возможность локализации приложения без опасности того, что элементы будут перекрывать друг друга.
- Визуализация применяет технологию DirectX и отличается высокой скоростью, извлекая крупные преимущества от аппаратного ускорения графики.
- Она предлагает надежную привязку к данным.
- Пользовательские интерфейсы могут быть описаны декларативно в XAML-файлах, которые поддерживаются независимо от файлов отделенного кода, что помогает отделить внешний вид от функциональности.

Однако размеры и сложность WPF обусловливают крутую кривую обучения.

Типы, предназначенные для написания WPF-приложений, находятся в пространстве имен `System.Windows` и всех его подпространствах за исключением `System.Windows.Forms`.

Windows Forms

Windows Forms — это API-интерфейс обогащенного клиента, который является ровесником самой платформы .NET Framework. По сравнению с WPF она является относительно простой технологией, которая предлагает большинство возможностей, необходимых во время разработки типового Windows-приложения. Она также играет важную роль в сопровождении унаследованных приложений. Тем не менее, в сравнении с WPF инфраструктура Windows Forms обладает рядом недостатков.

- Позиции и размеры элементов управления задаются в пикселях, что приводит к риску некорректного отображения приложений на клиентах с настройками DPI, отличающимися от таких настроек у разработчика (хотя в версии .NET Framework 4.7 ситуация несколько улучшилась).
- Для рисования нестандартных элементов управления используется API-интерфейс GDI+, который вопреки достаточно высокой гибкости медленно визуализирует крупные области (и без двойной буферизации может вызывать мерцание).
- Элементы управления лишены подлинной прозрачности.
- Большинство элементов управления не поддерживают компоновку. Например, поместить элемент управления изображением внутрь заголовка элемента управления вкладкой не удастся. Настройка списковых представлений и полей с раскрывающимися списками отнимает много времени и сил.
- Трудно добиться надежной работы динамической компоновки.

Последний пункт является веской причиной отдавать предпочтение WPF перед Windows Forms, даже если разрабатывается бизнес-приложение, которому требуется просто пользовательский интерфейс, а не учет "поведенческих особенностей пользователей". Элементы компоновки в WPF, подобные `Grid`, упрощают организацию меток и текстовых полей таким образом, что они будут всегда выровненными — даже при смене языка локализации — без запутанной логики и какого-либо мерцания. Кроме того, не придется приводить все к наименьшему общему знаменателю в смысле экранного разрешения — элементы компоновки WPF изначально проектировались с поддержкой изменения размеров.

В качестве положительного момента следует отметить, что инфраструктура Windows Forms относительно проста в изучении и все еще поддерживается в многочисленных элементах управления от независимых разработчиков.

Типы Windows Forms находятся в пространствах имен `System.Windows.Forms` (сборка *`System.Windows.Forms.dll`*) и `System.Drawing` (сборка *`System.Drawing.dll`*). Последнее пространство имен также содержит типы GDI+ для рисования специальных элементов управления.

Xamarin

Инфраструктура Xamarin, которой теперь владеет Microsoft, позволяет писать мобильные приложения на языке C#, которые ориентированы на iOS и Android, а также на Windows Phone. Будучи межплатформенной, она функционирует не только под управлением .NET Framework, но и в своей собственной среде (производной от среды с открытым кодом Mono). За дополнительной информацией обращайтесь на веб-сайт `https://www.xamarin.com`.

UWP (Universal Windows Platform)

Универсальная платформа Windows (UWP) предназначена для разработки приложений, ориентированных на настольные компьютеры и устройства Windows 10, которые распространяются через Windows Store. Ее API-интерфейс обогащенного клиента, на который большое влияние оказала инфраструктура WPF, рассчитан на построение пользовательских интерфейсов, управляемых касаниями, и для компоновки применяет язык XAML. Типы находятся в пространствах имен `Windows.UI` и `Windows.UI.Xaml`.

Silverlight

Платформа Silverlight также отделена от .NET Framework и позволяет создавать графические пользовательские интерфейсы, которые функционируют в веб-браузере, во многом похоже на Macromedia Flash. С развитием HTML5 в Microsoft перестали уделять внимание Silverlight.

Технологии серверной части

ADO.NET

ADO.NET — это управляемый API-интерфейс доступа к данным. Хотя название включает наименование используемой в 1990-х годах технологии ADO (ActiveX Data Objects — объекты данных ActiveX), ADO.NET — совершенно другая технология. Она содержит два основных низкоуровневых компонента.

Уровень поставщиков

Модель поставщиков определяет общие классы и интерфейсы для низкоуровневого доступа к поставщикам баз данных. Такие интерфейсы состоят из подключений, команд, адаптеров и средств чтения (однонаправленных курсоров, предназначенных только для чтения базы данных). Инфраструктура .NET Framework поставляется с собственной поддержкой Microsoft SQL Server, но доступно множество драйверов от независимых разработчиков для других баз данных.

Модель `DataSet`

`DataSet` — это структурированный кеш данных. Он напоминает примитивную базу данных в памяти, которая определяет такие SQL-конструкции, как таблицы, строки, столбцы, отношения и представления. За счет программирования с участием кеша данных можно сократить количество обращений к серверу, улучшая показатели масштабируемости сервера и отзывчивости пользовательского интерфейса обогащенного клиента. Объекты `DataSet` поддерживают сериализацию и могут передаваться по сети между клиентскими и серверными приложениями.

Поверх уровня поставщиков находятся три API-интерфейса, которые предлагают возможность запрашивать базы данных с помощью LINQ:

- Entity Framework (только .NET Framework);
- Entity Framework Core (.NET Framework и .NET Core);
- LINQ to SQL (только .NET Framework).

Все три технологии включают *объектно-реляционные отображатели* (object/relational mapper — ORM), которые автоматически отображают объекты (основанные на определяемых вами классах) на строки в базе данных. Это позволяет запрашивать такие объекты с применением LINQ (вместо написания SQL-операторов `SELECT`) и обновлять их без написания вручную SQL-операторов `INSERT/DELETE/UPDATE`. В результате сокращается объем кода внутри уровня доступа к данным в приложении (особенно вспомогательного кода) и обеспечивается строгая статическая безопасность типов. Указанные технологии также устраняют необходимость в наличии `DataSet` как вместилищ данных, хотя объекты `DataSet` по-прежнему предлагают уникальную возможность по хранению и сериализации изменений состояния (то, что особенно полезно в многоуровневых приложениях). В сочетании с `DataSet` можно использовать Entity Framework или LINQ to SQL, хотя такой подход несколько грубый ввиду неуклюжести самих `DataSet`. Другими словами, пока еще не существует прямолинейного готового решения для написания *n*-уровневых приложений с ORM от Microsoft.

Технология LINQ to SQL проще и быстрее Entity Framework, к тому же исторически производит лучший SQL-код (хотя Entity Framework улучшается благодаря многочисленным обновлениям). Технология Entity Framework более гибкая в том, что позволяет создавать точные отображения между базой данных и запрашиваемыми классами (Entity Data Model — модель сущностных данных), и предлагает модель, которая делает возможной независимую поддержку для баз данных, отличающихся от SQL Server.

Инфраструктура Entity Framework Core (EF Core) — это переписанная инфраструктура Entity Framework с более простым проектным решением, навеянным LINQ to SQL. Она отказывается от сложной модели сущностных данных и функционирует под управлением .NET Framework и .NET Core.

Стандарт .NET Standard 2.0 включает общие интерфейсы из уровня поставщиков, а также объекты `DataSet`, но исключает типы, специфичные для SQL Servers, и объектно-реляционные отображатели.

Windows Workflow (только .NET Framework)

Windows Workflow — это инфраструктура для моделирования и управления потенциально длительно выполняющимися бизнес-процессами. Инфраструктура Windows Workflow ориентирована на стандартную библиотеку времени выполнения, обеспечивая согласованность и возможность взаимодействия. Кроме того, она помогает сократить объем кодирования для динамически управляемых деревьев принятия решений.

Инфраструктура Windows Workflow не является строго серверной технологией — ее можно применять где угодно (например, организовать поток страницы в пользовательском интерфейсе).

Первоначально инфраструктура Windows Workflow появилась в версии .NET Framework 3.0, а ее типы были определены в пространстве имен `System.WorkFlow`. В .NET Framework 4.0 она была полностью пересмотрена; новые типы теперь находятся в пространстве имен `System.Activities`.

COM+ и MSMQ (только .NET Framework)

Инфраструктура .NET Framework позволяет взаимодействовать с COM+ для таких служб, как распределенные транзакции, через типы из пространства имен `System.EnterpriseServices`. Она также поддерживает технологию MSMQ (Microsoft Message Queuing — организация очереди сообщений Microsoft) для асинхронного однонаправленного обмена сообщениями посредством типов из пространства имен `System.Messaging`.

Технологии распределенных систем

Windows Communication Foundation (WCF)

WCF представляет собой сложную инфраструктуру для коммуникаций, которая появилась в версии .NET Framework 3.0. Она является гибкой и достаточно конфигурируемой для того, чтобы сделать своих предшественников — Remoting и Web Services (.ASMX) — *по большей части* излишними.

WCF, Remoting и Web Services похожи между собой в том, что все они реализуют описанную ниже базовую модель коммуникаций между клиентским и серверным приложениями.

- На стороне сервера вы указываете, какие методы могут быть вызваны удаленными клиентскими приложениями.
- На стороне клиента вы указываете или выводите *сигнатуры* серверных методов, которые должны вызываться.
- На сторонах сервера и клиента вы выбираете транспортный и коммуникационный протокол (в WCF это делается посредством *привязки*).
- Клиент устанавливает подключение к серверу.
- Клиент вызывает удаленный метод, который прозрачно выполняется на сервере.

Инфраструктура WCF еще более развязывает клиент и сервер через контракты служб и контракты данных. Концептуально вместо того, чтобы напрямую вызывать удаленный *метод*, клиент отправляет сообщение (XML или двоичное) конечной точке

удаленной *службы*. Одно из преимуществ такой развязки заключается в том, что клиенты не имеют какой-либо зависимости от платформы .NET или от любых патентованных коммуникационных протоколов.

Инфраструктура WCF исключительно конфигурируема и предлагает широкую поддержку для стандартизированных протоколов обмена сообщениями, основанных на SOAP (Simple Object Access Protocol — простой протокол доступа к объектам), в том числе расширения WS-* для безопасности. В результате появляется возможность взаимодействовать с участниками, выполняющими другое программное обеспечение — вероятно, на разных платформах — и в то же время по-прежнему поддерживать расширенные средства вроде шифрования. Однако на практике сложность этих протоколов ограничивает их адаптацию другими платформами, и в настоящий момент наилучшим вариантом для обмена сообщениями с возможностью взаимодействия является архитектурный стиль REST поверх HTTP, который в Microsoft поддерживается посредством уровня Web API на основе ASP.NET.

Тем не менее, для коммуникации между системами .NET инфраструктура WCF предлагает более развитую сериализацию и улучшенные инструменты, чем доступные с API-интерфейсами REST. Она также потенциально быстрее, не привязана к HTTP и может использовать двоичную сериализацию.

Типы, предназначенные для организации коммуникаций с помощью WCF, находятся в пространстве имен `System.ServiceModel` и его подпространствах.

Web API

Инфраструктура Web API функционирует поверх ASP.NET/ASP.NET Core и архитектурно подобна API-интерфейсу MVC от Microsoft за исключением того, что она спроектирована для открытия доступа к службам и данным, а не веб-страницам. Преимущество инфраструктуры Web API перед WCF связано с тем, что она позволяет следовать популярным соглашениям REST поверх HTTP, предлагая несложное взаимодействие с широчайшим диапазоном платформ.

Внутренне реализации REST проще протоколов SOAP и WS-*, на которые WCF полагается в плане возможности взаимодействия. С точки зрения архитектуры API-интерфейсы REST более элегантны для систем со слабой связностью, построены на основе фактических стандартов и великолепно задействуют то, что уже предоставляет протокол HTTP.

Remoting и .ASMX Web Services (только .NET Framework)

Remoting и .ASMX Web Services являются предшественниками WCF. С появлением WCF технология Remoting стала почти излишней, а .ASMX Web Services — излишней целиком и полностью.

Оставшаяся ниша Remoting касается коммуникаций между доменами приложений внутри одного и того же процесса (глава 24). Технология Remoting ориентирована на приложения с сильной связностью. Типичным примером может служить ситуация, когда клиент и сервер представляют собой приложения .NET, написанные одной компанией (или компаниями, разделяющими общие сборки). Коммуникации обычно предусматривают обмен потенциально сложными специальными объектами .NET, которые инфраструктура Remoting сериализирует и десериализирует без необходимости во внешнем вмешательстве.

Типы для Remoting находятся в пространстве имен `System.Runtime.Remoting` или его подпространствах, а типы для Web Services — в пространстве имен `System.Web.Services`.