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 составляют платформу .NET.

Некоторые из типов .NET используются напрямую CLR и являются критически важными для среды управляемого размещения. Эти 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)).

На уровне выше этого находятся дополнительные типы, которые расширяют функциональность уровня CLR, предоставляя такие средства, как XML, работа в сети и LINQ. Они находятся в *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. Эта ключевая инфраструктура в значительной степени определяет контекст оставшихся глав настоящей книги.

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

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

В табл. 5.1 представлена история совместимости между каждой версией C#, CLR и .NET Framework. Версия C# 5.0 требует в качестве целевой платформы CLR 4.5, что несколько странно, т.к. это исправленная версия CLR 4.0. Это значит, что приложения, ориентированные на CLR 4.0, будут выполняться под управлением CLR 4.5, когда эта версия позже будет установлена; следовательно, в Microsoft приложили максимум усилий, чтобы обеспечить обратную совместимость.

В этой главе дается обзор всех ключевых областей .NET Framework, начиная с основных типов, рассмотренных в этой книге, 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 (SP1)	3.5
4.0	4.0	4.0
5.0	4.5 (исправленная версия CLR 4.0)	4.5

Сборки и пространства имен в .NET Framework *пересекаются*. Наиболее экстремальными примерами являются *mscorlib.dll* и *System.Core.dll*, в которых определены десятки пространств имен, причем ни одно из них не начинается с *mscorlib* или *System.Core*. Однако менее очевидные случаи являются более запутанными, например, типы в `System.Security.Cryptography`. Большинство типов в этом пространстве имен находится в *System.dll*, исключая небольшое количество типов, расположенных в *System.Security.dll*. На веб-сайте, посвященном этой книге, содержится полное отображение пространств имен .NET Framework на сборки (`http://www.albahari.com/nutshell/namespacereference.aspx`).

Что нового в .NET Framework 4.5

Ниже перечислены новые функциональные возможности .NET Framework 4.5.

- Обширная поддержка асинхронности через методы возврата из задач.
- Поддержка протокола сжатия ZIP (глава 15).
- Улучшенная поддержка HTTP через новый класс `HttpClient` (глава 16).
- Улучшение показателей производительности для сборки мусора и извлечения ресурсов сборки.
- Поддержка взаимодействия WinRT и API-интерфейсов для построения планшетных приложений в стиле Metro.

Появились также новый класс `TypeInfo` (глава 19) и возможность указания таймаутов при сопоставлении с регулярными выражениями (глава 26).

В области параллельных вычислений стала доступной новая библиотека по имени Dataflow, предназначенная для построения сетей в стиле поставщик/потребитель.

Внесены также усовершенствования в библиотеки WPF, WCF и WF (Workflow Foundation).

Многие ключевые типы определены в следующих сборках: *mscorlib.dll*, *System.dll* и *System.Core.dll*. Первая из них, *mscorlib.dll*, содержит типы, требуемые самой исполняющей средой; *System.dll* и *System.Core.dll* включают дополнительные типы, требуемые программистами. Причины разделения последних двух сборок — чисто исторические: когда шла работа над версией .NET Framework 3.5, в Microsoft старались сделать ее насколько возможно добавочной, поскольку она выполнялась в существующей среде CLR 2.0 (точно так же, как .NET Framework 4.5 является добавочной версией по отношению к .NET Framework 4.0).

Таким образом, почти все новые ключевые типы (такие как классы, поддерживающие LINQ) вошли в новую сборку, которую в Microsoft назвали *System.Core.dll*.

Что нового в .NET Framework 4.0

Ниже перечислены новые функциональные возможности .NET Framework 4.0.

- Новые ключевые типы: `BigInteger` (для произвольно больших чисел), `Complex` (комплексные числа) и кортежи (глава 6).
- Новая коллекция `SortedSet` (глава 7).
- Контакты кода, предназначенные для того, чтобы позволить методам взаимодействовать более надежно посредством взаимных обязательств и ответственности (глава 13).
- Прямая поддержка файлов, отображаемых в памяти (глава 15).
- Методы отложенного ввода-вывода файлов и каталогов, которые возвращают `IEnumerable<T>` вместо массивов (глава 15).
- Исполняющая среда динамического языка (Dynamic Language Runtime – DLR), которая теперь является частью .NET Framework (глава 20).
- Прозрачность безопасности, которая упрощает защиту библиотек в средах с частичным доверием (глава 21).
- Новые конструкции многопоточности, в том числе более надежная перегрузка `Monitor.Enter`, `Thread.Yield`, новые сигнальные классы (`Barrier` и `CountdownEvent`) и примитивы отложенной ("ленивой") инициализации (глава 22).
- API-интерфейсы параллельного программирования для многоядерных процессоров, включая Parallel LINQ (PLINQ), императивные конструкции распараллеливания данных и задач, параллельные коллекции, а также примитивы синхронизации с малым ожиданием и взаимоблокировка (глава 23).
- Методы для мониторинга ресурсов доменов приложений (глава 24).

В .NET Framework 4.0 также содержатся расширения ASP.NET, в том числе инфраструктуры MVC и Dynamic Data, а также расширения Entity Framework, WPF, WCF и Workflow. Вдобавок имеется библиотека Managed Extensibility Framework (инфраструктура управляемой расширяемости), предназначенная для поддержки композиции, обнаружения и внедрения зависимостей во время выполнения.

Среда CLR и ядро платформы

Системные типы

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

Эти 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 рассматриваться 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 это в главе 6.

Пространство имен `System.Text.RegularExpressions` содержит типы, которые выполняют расширенные операции поиска и замены на основе шаблона; эти 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, рассмотренных 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 к потокам (глава 23)
```

Запросы

Язык интегрированных запросов (Language Integrated Query — LINQ) появился в .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. Большое преимущество языка LINQ в том, что он представляет собой согласованный API-интерфейс запросов для разнообразных предметных областей. Типы для запросов LINQ находятся в перечисленных ниже пространствах имен:

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

Кроме того, полный профиль .NET включает и следующие пространства имен:

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

(Профиль Metro исключает все пространства имен `System.Data.*`.)

API-интерфейсы LINQ to SQL и Entity Framework используют низкоуровневые типы ADO.NET из пространства имен `System.Data`.

XML

Язык XML широко используется внутри .NET Framework. В главе 10 внимание сосредоточено целиком на LINQ to XML — облегченной объектной модели документа XML, которую можно сконструировать и опрашивать с помощью LINQ. В главе 11 описана старая модель W3C DOM, а также высокопроизводительные низкоуровневые

классы для чтения/записи и 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
```

Следующие пространства имен доступны в стандартных профилях .NET (но не в Metro):

```
System.Xml.XPath         // Язык запросов XPath
System.Xml.Xsl           // Поддержка стилевых таблиц
```

Диагностика и контракты кода

В главе 13 мы раскроем возможности .NET по регистрации в журнале и утверждениям, а также систему контрактов кода, которая появилась в .NET Framework 4.0. Мы покажем, как взаимодействовать с другими процессами, выполнять запись в журнал событий Windows и использовать счетчики производительности для проведения мониторинга. Типы для этих целей определены в пространстве имен `System.Diagnostics` и его подпространствах.

Параллелизм и асинхронность

Большинству современных приложений приходится иметь дело с более чем одной вещью, происходящей в один и тот же момент времени. Язык C# 5.0 и платформа .NET Framework 4.5 делают это проще, чем было ранее, через асинхронные функции и такие высокоуровневые конструкции, как задачи и комбинаторы задач. Все это подробно рассматривается в главе 14, которая начинается с объяснения 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

Платформа .NET Framework предоставляет потоковую модель для низкоуровневого ввода-вывода. Обычно для чтения и записи непосредственно в файлы и сетевые подключения используются потоки данных, которые могут быть соединены или помещены внутрь декорированных потоков для добавления функциональности сжатия или шифрования. В главе 15 рассматривается архитектура потоков .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 в пространстве имен `System.Net` можно напрямую работать со стандартными сетевыми протоколами, такими как HTTP, FTP, TCP/IP и SMTP. В главе 16 мы покажем, как организовать взаимодействие с помощью каждого из этих протоколов, начав с простых задач вроде загрузки веб-страницы и закончив использованием 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 доступны в приложениях Metro; вместо них должны применяться типы WinRT.

Сериализация

В .NET Framework предоставляется несколько систем для сохранения и восстановления объектов в бинарном или текстовом представлении. Такие системы являются обязательными в технологиях распределенных приложений, таких как WCF, Web Services и Remoting, а также используются для сохранения и восстановления объектов в файле. В главе 17 мы рассмотрим три механизма сериализации: сериализатор контракта данных, бинарный сериализатор и сериализатор XML. Типы, связанные с сериализацией, находятся в следующих пространствах имен:

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

В профиле Metro исключается механизм бинарной сериализации.

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 делать такие вещи, как динамический вызов методов. Посредством `Reflection.Emit` можно конструировать новый код на лету.

В главе 18 мы покажем, как компоновать и подписывать сборки, использовать глобальный кеш сборок и ресурсы, а также разрешать ссылки на файлы. В главе 19 мы опишем рефлексию и атрибуты, продемонстрировав, как просматривать метаданные, динамически вызывать функции, записывать специальные атрибуты, выдавать новые типы и проводить разбор низкоуровневого кода IL. Типы, предназначенные для рефлексии и работы со сборками, находятся в следующих пространствах имен:

```
System
System.Reflection
System.Reflection.Emit (в профиле Metro отсутствует)
```

Динамическое программирование

В главе 20 мы рассмотрим некоторые шаблоны для динамического программирования и работы со средой DLR, которая является частью CLR, начиная с .NET Framework 4.0. Мы покажем, как реализовать шаблон Visitor (Посетитель), записывать специальные динамические объекты и взаимодействовать с IronPython. Типы, связанные с динамическим программированием, находятся в пространстве имен `System.Dynamic`.

Безопасность

Платформа .NET Framework поддерживает собственный уровень безопасности, позволяя организовать работу в песочнице как для других сборок, так и для своей.

В главе 21 мы рассмотрим доступ кода, роли и идентичность, а также модель прозрачности, появившаяся в CLR 4.0. Затем мы раскроем криптографические возможности .NET Framework, рассмотрев шифрование, хеширование и защиту данных. Типы для этих целей определены в следующих сборках:

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

В профиле Metro доступно только пространство имен `System.Security`; криптографические возможности поддерживаются в WinRT.

Расширенная многопоточность

Асинхронные функции C# 5 значительно упрощают параллельное программирование, поскольку они уменьшают потребность в работе с низкоуровневыми технологиями. Тем не менее, все еще возникают ситуации, при которых нужны сигнальные конструкции, локальное хранилище потока, блокировки чтения/записи и т.п. Все эти вопросы подробно рассматриваются в главе 22. Типы, относящиеся к многопоточности, находятся в пространстве имен `System.Threading`.

Параллельное программирование

В главе 23 мы детально рассмотрим библиотеки и типы для работы с многоядерными процессорами, включая API-интерфейсы для 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 рассмотрим 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, как применять Remoting для взаимодействия с доменами приложений. Тип `AppDomain`, определенный в пространстве имен `System`, применим только к приложениям, отличным от Metro.

Собственная возможность взаимодействия и взаимодействие с COM

Существует возможность взаимодействия как с собственным кодом, так и с кодом COM. Собственная возможность взаимодействия позволяет вызывать функции из неуправляемых DLL-библиотек, регистрировать обратные вызовы, отображать структуры данных и взаимодействовать с собственными типами данных. Возможность взаимодействия с COM позволяет обращаться к типам COM и открывать типы .NET для COM. Типы, поддерживающие все это, находятся в пространстве имен `System.Runtime.InteropServices`; мы рассмотрим их в главе 25.

Прикладные технологии

Технологии пользовательских интерфейсов

Платформа .NET Framework предлагает четыре API-интерфейса для применения при построении приложений с пользовательским интерфейсом.

- *ASP.NET* (`System.Web.UI`). Предназначен для написания приложений тонких клиентов, которые выполняются в стандартном веб-браузере.
- *Silverlight*. Предназначен для построения расширенных пользовательских интерфейсов внутри веб-браузера.
- *Windows Presentation Foundation* (`System.Windows`). Предназначен для написания приложений обогащенных клиентов.
- *Windows Forms* (`System.Windows.Forms`). Предназначен для поддержки унаследованных приложений обогащенных клиентов.

В общем случае приложение тонкого клиента сводится к веб-сайту; приложение обогащенного клиента — это программа, которую конечный пользователь должен загружать или устанавливать на своем компьютере.

ASP.NET

Приложения, написанные с использованием ASP.NET, располагаются на сервере Windows IIS и могут быть доступны с помощью почти всех веб-браузеров. Ниже перечислены преимущества ASP.NET по сравнению с технологиями обогащенных клиентов.

- Отсутствие потребности в развертывании на клиентской стороне.
- Клиенты могут использовать платформы, отличные от Windows.
- Простое развертывание обновлений.

Кроме того, поскольку бо́льшая часть кода, который приходится писать в приложении ASP.NET, выполняется на сервере, уровень доступа к данным проектируется для выполнения в том же самом домене приложения — без ограничения безопасности или масштабируемости. В противоположность этому, обогащенный клиент, который делает то же самое, в общем случае не настолько безопасен или масштабируем. (Решением для обогащенного клиента является вставка *среднего уровня* между клиентом и базой данных. Этот 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 можно выбирать между традиционным API-интерфейсом Web Forms и 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. Инфраструктура MVC появилась в .NET Framework 4.0 и с тех пор обрела зрелость. В целом она предоставляет лучшую программную абстракцию, чем Web Forms; она также позволяет иметь больший контроль над генерируемой HTML-разметкой. Однако есть один аспект, в котором MVC проигрывает Web Forms — визуальный конструктор. Это сохраняет Web Forms в качестве хорошего средства для построения веб-страниц с преимущественно статическим содержимым.

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

- интерфейс веб-браузера существенно ограничивает то, что можно делать;
- поддержка состояния на стороне клиента (или от имени клиента) является громоздкой.

Тем не менее, интерактивность и отзывчивость можно улучшить с помощью сценариев клиентской стороны или технологий наподобие AJAX: хорошим ресурсом для этого является веб-сайт `http://ajax.asp.net`. Работа с AJAX упрощается за счет использования таких библиотек, как jQuery.

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

Silverlight

Формально Silverlight не является частью .NET Framework: это отдельная платформа, которая содержит подмножество ключевых средств .NET Framework, а также поддерживает возможность выполнения в виде подключаемого модуля веб-браузера. Графическая модель Silverlight — в сущности, подмножество WPF и это позволяет применять существующие знания при разработке Silverlight-приложений. Подключаемый модуль Silverlight доступен как межплатформенный загружаемый файл для веб-браузером, что очень похоже на Macromedia Flash.

Flash обладает намного большей установочной базой, поэтому доминирует в данной области. По этой причине Silverlight, как правило, используется в пограничных сценариях, например, в корпоративных сетях.

Metro

Windows-библиотека Metro не является частью .NET Framework и предназначена для разработки планшетных пользовательских интерфейсов в Windows 8 (см. раздел "C# и Windows Runtime" в главе 1). API-интерфейс Metro реализован в духе WPF и использует для компоновки язык XAML. Поддерживающие пространства имен — `Windows.UI` и `Windows.UI.Xaml`.

Windows Presentation Foundation (WPF)

Инфраструктура WPF появилась в .NET Framework 3.0 и предназначена для написания приложений обогащенных клиентов. Ниже перечислены преимущества WPF по сравнению с Windows Forms.

- Она поддерживает развитую графику, включая произвольные трансформации, трехмерную визуализацию и истинную прозрачность.
- Первичная единица измерения основана не на пикселях, поэтому приложения корректно отображаются при любой настройке DPI (dots per inch — точек на дюйм).
- Она имеет обширную поддержку динамической компоновки, которая означает возможность локализации приложения без опасности того, что элементы будут перекрывать друг друга.
- Визуализация использует DirectX и является быстрой, получая преимущества от аппаратного ускорения графики.

- Пользовательские интерфейсы могут быть описаны декларативно в XAML-файлах, которые поддерживаются независимо от файлов отделенного кода — это помогает отделить внешний вид от функциональности.

Тем не менее, размеры и сложность WPF предполагают крутую кривую обучения.

Типы для написания WPF-приложений находятся в пространстве имен `System.Windows` и всех его подпространствах кроме `System.Windows.Forms`.

Windows Forms

Windows Forms — это API-интерфейс обогащенного клиента, который является ровесником .NET Framework. По сравнению с WPF это относительно простая технология, предлагающая большинство возможностей, необходимых при написании типового Windows-приложения. Она также важна для поддержки унаследованных приложений. Однако если сравнивать с WPF, 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 настройки у разработчиков.
- API-интерфейсом для рисования нестандартных элементов управления является GDI+, который, несмотря на достаточную гибкость, медленно визуализирует крупные области (и без двойной буферизации может привести к мерцанию).
- У элементов управления отсутствует истинная прозрачность.
- Трудно добиться надежности динамической компоновки.

Последний пункт является отличной причиной отдать предпочтение WPF перед Windows Forms, даже если разрабатывается бизнес-приложение, которому требуется просто пользовательский интерфейс, а не "опыт взаимодействия". 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 неаккуратной логики и мерцания. Кроме того, не придется приводить все к наименьшему общему знаменателю в плане 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 по-прежнему широко поддерживается в элементах управления третьих сторон.

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

Технологии серверной части

ADO.NET

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

Уровень поставщиков

Модель поставщиков определяет общие классы и интерфейсы для низкоуровневого доступа к поставщикам баз данных. Эти интерфейсы состоят из подключений, команд, адаптеров и средств чтения (однонаправленных курсоров, предназначенных только для чтения, в базе данных). Платформа .NET Framework поставляется с собственной поддержкой Microsoft SQL Server и Oracle, а также имеет поставщики OLE-DB и ODBC.

Модель DataSet

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

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

- LINQ to SQL (появился в .NET Framework 3.5);
- Entity Framework (появился в .NET Framework 3.5 SP1).

Обе 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 на уровне доступа к данным в приложении (особенно вспомогательного кода) и обеспечивается строгая безопасность статических типов. Эти технологии также устраняют потребность в наличии `DataSet` в качестве вместилищ данных, хотя `DataSet` по-прежнему предлагают уникальную возможность по хранению и сериализации изменений состояния (то, что особенно полезно для многоуровневых приложений). Совместно с `DataSet` можно использовать LINQ to SQL или Entity Framework, хотя этот процесс несколько топорный ввиду неуклюжести самих `DataSet`. Другими словами, пока что не существует очевидного готового решения для написания *n*-уровневых приложений с ORM от Microsoft.

LINQ to SQL проще и быстрее Entity Framework, к тому же производит лучший SQL-код (несмотря на усовершенствования Entity Framework в последних версиях). Технология Entity Framework более гибкая в том, что позволяет создавать точные отображения между базой данных и запрашиваемыми классами, и предлагает модель, которая допускает поддержку от третьих сторон для баз данных, отличных от SQL Server.

Windows Workflow

Windows Workflow — это инфраструктура для моделирования и управления потенциально долго выполняющимися бизнес-процессами. Рабочий поток нацелен на стандартную библиотеку времени выполнения, предоставляя согласованность и возможность взаимодействия. Рабочий поток также помогает сократить объем кодирования для динамически управляемых деревьев принятия решений.

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

Концепция рабочего потока первоначально появилась в версии .NET Framework 3.0, с типами, определенными в пространстве имен `System.WorkFlow`. В .NET Framework 4.0 эта концепция была полностью пересмотрена; новые типы теперь находятся в пространстве имен `System.Activities`.

COM+ и MSMQ

Платформа .NET Framework позволяет взаимодействовать с COM+ для таких служб, как распределенные транзакции, через типы в пространстве имен `System.EnterpriseServices`. Она также поддерживает 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 излишними своих предшественников — Remoting и Web Services (.ASMX).

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 через привязку).
- Клиент устанавливает подключение к серверу.
- Клиент вызывает удаленный метод, который прозрачно выполняется на сервере.

WCF еще более развязывает клиент и сервер посредством контрактов служб и контрактов данных. Концептуально вместо того, чтобы напрямую вызывать удаленный *метод*, клиент отправляет сообщение (XML или бинарное) конечной точке удаленной *службы*. Одно из преимуществ такой развязки заключается в том, что клиенты не имеют никаких зависимостей от платформы .NET либо от любых патентованных коммуникационных протоколов.

Инфраструктура WCF является исключительно конфигурируемой и предлагает наиболее широкую поддержку для стандартизированных протоколов обмена сообщениями, включая WS-*. Это позволяет взаимодействовать с участниками, выполняющими другое программное обеспечение — возможно, на разных платформах, — и в то же время поддерживать расширенные средства вроде шифрования. Еще одно преимущество WCF состоит в том, что вы можете изменять протоколы без потребности в изменении других аспектов клиентского или серверного приложения.

Типы, имеющие отношение к WCF, находятся в пространстве имен `System.ServiceModel`.

Remoting и .ASMX Web Services

Remoting и .ASMX Web Services — это предшественники WCF, которые с появлением WCF стали в основном излишними, хотя Remoting все еще используется в коммуникациях между доменами приложений внутри одного и того же процесса (см. главу 24).

Функциональность Remoting ориентирована на приложения с сильной связностью. Типичным примером может служить ситуация, когда и клиент, и сервер являются приложениями .NET, написанными одной компанией (или компаниями, разделяющими общие сборки). Коммуникации обычно предусматривают обмен потенциально сложными специальными объектами .NET, которые инфраструктура Remoting сериализирует и десериализирует безо всякого вмешательства извне.

Функциональность Web Services ориентирована на приложения со слабой связностью или приложения в стиле SOA. Типичным примером может служить ситуация, когда сервер спроектирован на прием простых SOAP-сообщений, которые инициируются клиентами, выполняющими разное программное обеспечение, возможно, на различных платформах. Инфраструктура Web Services может использовать только HTTP и SOAP в качестве транспортных и форматирующих протоколов, а приложения обычно размещаются на сервере IIS. За преимущества взаимодействия приходится платить снижением производительности — приложение Web Services обычно медленнее, как при выполнении, так и при разработке, чем хорошо спроектированное приложение Remoting.

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

CardSpace

CardSpace — это протокол аутентификации на основе маркеров и управления идентичностью, разработанный с целью упрощения управления паролями для конечных пользователей. Этой технологии было уделено не особенно большое внимание из-за трудностей в переносе маркеров между машинами (популярной альтернативой, лишенной этой проблемы, является *OpenID*).

Технология CardSpace построена на открытых стандартах XML, поэтому в ней могут принимать участие стороны, не зависящие от Microsoft. Пользователь может иметь несколько идентичностей, которые управляются третьей стороной (*поставщик идентичностей*). Когда пользователь желает получить доступ к ресурсу на сайте X, он аутентифицируется для поставщика идентичностей, который затем выдает маркер для сайта X. Это устраняет необходимость предоставления пароля непосредственно сайту X и сокращает количество идентичностей, которыми пользователь должен управлять.

WCF позволяет указывать идентичность CardSpace при подключении через защищенный канал HTTP с помощью типов из пространств имен `System.IdentityModel.Claims` и `System.IdentityModel.Policy`.